Mobile Ad Hoc Network Naming System

Gonçalo Gonçalves
Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2017

Orientador
Pedro Brandão, Professor Auxiliar, Faculdade de Ciências
Coorientador
Rui Prior, Professor Auxiliar, Faculdade de Ciências

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, / /

Abstract

The dissemination of the Wi-Fi technology since the early 2000s, led to the rising popularity of mobile computing devices such as laptops or smartphones. These devices are usually connected to each other by way of an access point/router, which configures and manages communication between them. In situations where no infrastructure is available, mobile devices are still able to communicate using the ad hoc wireless networking mode, allowing them to establish an impromptu peer-to-peer network (i.e., a Mobile Ad Hoc Network (MANET)).

This dissertation focuses on how to perform name resolution in a MANET. Because the Domain Name System (DNS) heavily relies on a fixed and pre-configured infrastructure of name servers, it is not an option in an ad hoc environment. Therefore, a new system is needed to support name resolution.

To that end, we researched what had already been done in this field and then formulated a solution of our own. We designed two name resolution protocols: (1) Naming Extension for AODV (NExtA), based on including name resolution in the route discovery process of a reactive routing protocol and (2) Name System for Pro-active Routing-Enabled Ad Hoc Networks (nameSPREAD), for pro-active routing networks, based on the concept of always asking the next hop for the name of the destination of a newly learned route.

We implemented nameSPREAD and tested it (using Optimized Link State Routing (OLSR) as the routing protocol) in different scenarios with varying numbers of nodes and network densities. The results were promising, seeing that the protocol was effective in providing the nodes with each other's names while introducing very little overhead over the routing protocol. We concluded that nameSPREAD would be a viable option for name resolution in mobile ad hoc networks.

Resumo

A disseminação da tecnologia Wi-Fi desde o início dos anos 2000, levou à crescente popularidade de dispositivos computacionais móveis, como computadore portáteis ou smartphones. Tipicamente, estes dispositivos são ligados uns aos outros por meio de um *access point*/*router*, que configura e gere a comunicação entre os mesmos. Em situações em que nenhuma infraestrutura está disponível, os dispositivos móveis podem comunicar usando o *ad hoc mode* das redes sem fio, permitindo estabelecer uma rede *peer-to-peer* improvisada (ou seja, uma Mobile Ad Hoc Network (MANET)).

Esta dissertação foca-se em como executar resolução de nomes em redes MANET. Como o Domain Name System (DNS) depende fortemente de uma infra-estrutura fixa e pré-configurada de servidores de nomes, não é uma opção num ambiente *ad hoc*. Portanto, um novo sistema é necessário para suportar a resolução de nomes.

Para esse fim, pesquisámos o que já foi feito neste campo e depois formulámos uma solução própria. Criámos dois protocolos de resolução de nomes: (1) Naming Extension for AODV (NExtA), com base na inclusão da resolução de nomes no processo de descoberta de rota dos protocolos de encaminhamento reativos e (2) Name System for Pro-active Routing-Enabled Ad Hoc Networks (nameSPREAD), para redes de encaminhamento pro-ativo, com base no conceito de pedir sempre ao próximo salto o nome do destino de qualquer rota recém-descoberta.

Implementámos o nameSPREAD e testámo-lo (usando Optimized Link State Routing (OLSR) como o protocolo de encaminhamento) em diferentes cenários com diferentes números de nós e densidades de rede. Os resultados foram promissores, visto que o protocolo foi eficaz em fornecer aos nós os nomes uns dos outros, apresentando um *overhead* muito curto sobre o protocolo de encaminhamento. Concluímos que o nameSPREAD seria uma opção viável para resolução de nomes em redes *ad hoc* móveis.

Keywords

manet, name resolution, proactive routing, reactive routing

Contents

List of Tables

List of Figures

Code Blocks

Acronyms

AA Address Agent

ADNS MANET Distributed Naming System

ANS Ad Hoc Name Service System

ANS Ad Hoc Name Service System

AODV Ad hoc On-demand Distance Vector

AP Access Point

API Application Programming Interface

ARP Address Resolution Protocol

DHCP Dynamic Host Configuration Protocol

DHT Distributed Hash Table

DNS-SD DNS Service Discovery

DNS Domain Name System

DoS Denial of Service

DSDV Destination-Sequenced Distance-Vector

DSR Dynamic Source Routing

ESP Encapsulating Security Payload

FIFO first in, first out

FQDN Fully Qualified Domain Name

HID Host ID

HMAC Hash-based Message Authentication Code

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

IPSec Internet Protocol Security

IP Internet Protocol

JSON JavaScript Object Notation

LLMNR Link-Local Multicast Name Resolution

LUNARng Lightweight Underlay Network Ad hoc Routing next generation

LUNAR Lightweight Underlay Network Ad hoc Routing

MANET Mobile Ad Hoc Network

MAODV Multicast Ad Hoc On-demand Distance Vector

mDNS Multicast DNS

MNS MANET Naming Service

MOSS Multicast Onomastic Search

NA Not Available

NACK Negative Acknowledgement

nameSPREAD Name System for Pro-active Routing-Enabled Ad Hoc Networks

NC Name Conflict

NDR Name Directory Service

ND Neighbor Discovery

NExtA Naming Extension for Ad hoc On-demand Distance Vector (AODV)

NE Named Entity

NREP Name Resolution Reply

NREQ Name Resolution Request

NSS Name Service Switch

NS Name Server

ns network simulator

OLSR Optimized Link State Routing

OONF OLSR.org Network Framework

PGM Pragmatic General Multicast

PID Partition ID

PNR Pending Name Requests

QoS Quality of Service

RERR Route Error

RFC Request for Comments

RREP Route Reply

RREQ Route Request

TC Traffic Control

TCP Transmission Control Protocol

TBRPF Topology Broadcast Based on Reverse-Path Forwarding

TORA Temporally Ordered Routing Algorithm

TTL Time to Live

UDP User Datagram Protocol

UE User Entity

VANETs Vehicular Ad Hoc Networks

WRP Wireless Routing Protocol

WSNs Wireless Sensor Networks

Chapter 1

Introduction

In the early 2000s, mobile computing devices started to achieve widespread adoption and came to be perceived as useful for everyday use and not just for work. This high demand was mainly due to two reasons. First, the natural decrease of market prices of the technologies involved in producing such devices and second, the introduction of the Wi-Fi technology. The latter was decisive in the popularization of devices such as laptops and smartphones, because it allowed them to connect to each other and — perhaps more importantly — to the Internet.

The most common way to allow these wireless devices to communicate with each other is to have them be connected to an Access Point (AP)/router which will configure the devices' network attributes and manage communication between them. However, there might be situations in which no pre-configured infrastructure is available. To allow communication in such situations, the ad hoc wireless networking mode can be used. This mode provides wireless devices with a way to establish a peer-to-peer network, which can then be extended to provide multi-hop communication between them and to manage itself in terms of addressing and routing (see Section 2.1.1). When this kind of network is created by devices that are especially mobile, we call it a Mobile Ad Hoc Network (MANET) [7, p. 76].

MANETs are specially appealing in situations where multiple mobile nodes must communicate but, for some reason, they cannot rely on traditional wired or cellular infrastructure, which might be the case for soldiers in a battlefield, police search/rescue operations, firemen squads or disaster scenarios in which communication base stations get destroyed [41, p. 2].

Other suitable uses for MANETs include data gathering in mobile Wireless Sensor Networks (WSNs) [7, p. 492,493] and inter-vehicle communication for Vehicular Ad Hoc Networks (VANETs) [7, p. 117].

There have been many proposed protocols for MANETs (see [25] and Chapter 3) but few have reached Request for Comments (RFC) status and most are routing related.

This work addresses a different service, one that is fundamental in infrastructured networks, namely in the Internet, but traditionally absent in MANETs — name resolution.

1.1 Problem Description and Motivation

A naming system is an important part of any network. It allows us to communicate with other machines by knowing only their names, instead of having to memorize Internet Protocol (IP) addresses that are several bytes long and are, therefore, not suitable for frequent use by humans. This is why we use the Domain Name System (DNS) in the Internet, otherwise it would not be as widespread as it is today.

This notion must also be taken into account when dealing with MANETs. However, we cannot use DNS, because it heavily relies on a fixed and pre-configured infrastructure of name servers — something that is not possible in a MANET (see Section 2.3.1). This lack of infrastructure, coupled with the mobility of the devices and the power, processing and bandwidth limitations they usually have, make it very difficult to create an efficient naming system.

1.2 Objectives

The purpose of this dissertation is to study the existing proposals for name resolution in MANETs (see Chapter 3) and examine their benefits and shortcomings, in order to develop a new protocol that can effectively and efficiently perform the tasks (defined in Section 2.3.2.2) that must be expected of a MANET naming system. In addition to being effective in providing names to the nodes in a MANET, the protocol should be designed in such a way that it can perform name resolution while imposing the least possible message overhead in relation to the underlying protocols (i.e., the routing protocol).

Due to practical reasons, it might only be possible to test the developed protocol with virtual network simulation software like OMNeT++ [59] or network simulator (ns) version 2 or 3 [22].

1.3 Dissertation Outline

The remaining chapters of this dissertation are organized as follows.

In Chapter 2 we discuss relevant information for the comprehension of this dissertation, such as: the concept of MANET and characteristics/challenges associated with it; a brief description of routing protocols for ad hoc networks; a comparison of possible architectures for a MANET naming system and the requirements it should be able to meet. Chapter 3 exposes a survey of the most significant proposals for name resolution in MANETs, with an analysis of their benefits and flaws.

In Chapter 4, we present our own proposals of a name resolution system for ad hoc networks with either pro-active or reactive routing protocols. This is followed by Chapter 5, in which we describe the actual implementation of one of our proposed naming systems.

After having detailed the design and implementation of our protocol, we proceed to Chapter 6, where we establish different testing scenarios and then present the results of running the protocol in said scenarios.

Finally, we conclude this dissertation with Chapter 7, by summarising the goals that where accomplished and propose the remaining ones as future work.

Chapter 2

Background

For a better understanding of what we will discuss in the following chapters, we need some familiarity with the basic concepts behind Mobile Ad Hoc Networking. To that end, this chapter describes the key aspects that make Mobile Ad Hoc Networks different from conventional infrastructure-based networks (in which nodes communicate via router or Access Point (AP)) and what difficulties are to be expected with respect to their limited resources and their dynamic nature.
We then proceed to a brief explanation of how routing can be performed in Mobile Ad Hoc Networks, followed by a description of the general requirements a naming system should be able to meet and the architectural options one must consider when dealing with name resolution in such networks.

2.1 Mobile Ad Hoc Networks

Ad hoc is a Latin expression that literally means "for this" [26]. Nowadays, when we describe something as being *ad hoc*, we are generally referring to something that was improvised for a particular purpose and will be discarded once that purpose is fulfilled.

Accordingly, an **ad hoc network** is an on-the-fly, improvised and autonomous wireless network that connects nodes in a peer-to-peer fashion, without the need for a pre-configured infrastructure. While conventional networks require a router or AP to which all nodes are connected and through which all communication will go, ad hoc networks rely on the nodes themselves to do all the work, as illustrated in Figure 2.1. Nodes in the network must rely on each other for basic functional operations, like the routing (see 2.2) and forwarding of packets [41, p. 2]. Furthermore, each connection between nodes is independent of the others in the network, so if that connection fails, the other connections continue to function.

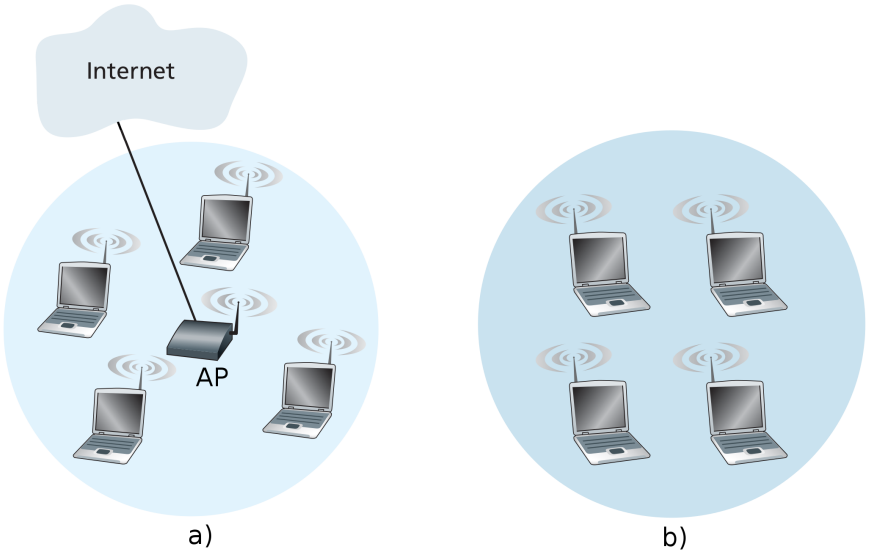

Figure 2.1: a) Infrastructure-based network; b) Ad hoc network [32, p. 527]

In such a network, if the nodes do not have fixed positions and are able to move at will (along with other characteristics that we explore in 2.1.1), we call it a **Mobile Ad Hoc Network (MANET)**, which can be composed of laptops, smartphones or any mobile device with a wireless connection. As a result of not having the need for an infrastructure and because they are expecting nodes to move freely, MANETs can be specially appealing in situations where multiple mobile nodes must quickly establish a way to communicate but, for some reason, they cannot rely on traditional wired or cellular infrastructure, as might be the case with the military, the police, firemen squads or any kind of operation that takes place in a remote location.

We will now explore the most relevant characteristics of MANETs that set them apart from other, more traditional, types of networks. Some of these characteristics can be thought of as limiting factors and so, in the interest of clarity, they are divided in two different subsections.

2.1.1 Main Characteristics

As mentioned in 2.1, a significant aspect in MANETs is that nodes may not have a stable position, they are free to move around at any given time, which means that the network protocols must be flexible enough to adapt to a **dynamic topology**. These nodes can take the form of all sorts of different devices which can be using different systems and have different resources. This can be another feature of MANETs: **system/device heterogeneity** [7, p. 77], which does not affect the network, as it should work regardless of the types of devices/systems that constitute it.

By default, a node in an ad hoc network, is only able to communicate with its neighbors, that is, nodes in the single-hop reach area [52, p. 1]. This is a big drawback because it restricts communication to the node's radio range. To solve this problem, MANETs use specially designed routing protocols that provide the network with **multi-hop communication** (see 2.2). A path between two nodes is therefore a sequence of an arbitrary number of hops of intermediate nodes.

Perhaps the most important aspect of an ad hoc network is its capability to be autonomous. This is, in fact, a necessity warranted by the lack of a centralized entity to manage communication and configuration of nodes. Infrastructure-based networks are not autonomous. They need to be pre-configured (i.e. with a Dynamic Host Configuration Protocol (DHCP) server) in order to provide new nodes with IP addresses for the default gateway, the name server and for the nodes themselves. MANET nodes cannot rely on a server from which to request these parameters, so they must rely on each other for addressing, routing, clustering, etc [41, p. 4]. We refer to this self-reliance ability as **self-configuration**. This also implies that MANETs are networks of **quick deployment** since they do not need planning or extra hardware, like routers, switches or cables.

2.1.2 Main Challenges

When working with MANETs, we should also be aware of their shortcomings. Regardless of features that might be considered advantageous, MANETs are not without problems because those same features can become obstacles that should be taken into account.

Following the earlier argument that the dynamic nature of the topology of MANETs makes them flexible — which is useful from the point of view of a node — we will now consider it from another perspective: that of the network itself. Nodes are free to wander around and in doing so, they can come in and out of radio range of the MANET. Coupled with the usual **wireless medium issues** (path loss, multipath fading, noise, etc) [5, p. 118], this will cause an **intermittent connection** which will lead to a series of problems that can be difficult to solve. For instance, the **routing overhead** will increase because the nodes will have to exchange a high quantity of messages trying to keep up with constant topology changes (see 2.2), which can severely affect performance and consequently debilitate Quality of Service (QoS).

Another problem that arises from the mobility of nodes is **network partitioning/mergence**. Partitioning happens when a connection between two nodes that were bridging different clusters of nodes, is lost. This results in the partition of the network into two different MANETs, which will render communication impossible between them. Conversely, mergence happens when different

MANETs are suddenly linked by two nodes (one from each network) that come in radio range of each other. This can cause problems due to nodes from each original MANET having the same address or name [44, p. 6]. A MANET must therefore have a routing protocol that can detect when nodes leave or join the network and addressing and naming protocols that are able to detect and resolve address and name conflicts.

MANET nodes will typically be battery powered mobile devices, which can have very **limited resources** [41, p. 4] like CPU processing power, memory, bandwidth or battery autonomy. Protocols must take these limitations into account and try to perform their functions in the most efficient way possible because if the nodes rely on each other, then the network is only as good as its weakest link.

Typically, MANETs are thought of as flat distributed networks whose operation depends on every node equally (see Section 2.3.2.1 for different architectural approaches). While in most small networks that will certainly be the case, it is not that clear when the network grows to a considerable size. Regular infrastructure-based networks can be scaled by using the inherent hierarchy of the network topology. Because MANETs do not rely on an infrastructure (at least not natively), this is not an option. Consequently, they are usually reduced to the use of protocols which heavily rely on flooding techniques and thus impose an excessive amount of overhead on the network. **Scalability** is therefore one of the most difficult challenges [41, p. 4] of working with MANETs because it is very hard to achieve efficiently in a mobile environment.

One last issue we should point out is the matter of **security and privacy**. As wireless networks, MANETs suffer from the well known vulnerabilities of wireless communication [41, p. 5]. Because they use a shared medium for all nodes, it becomes easier for attackers to eavesdrop on communications. The intricacy of ad hoc protocols and the lack of a central managing entity, makes it even harder to detect attacks on the network.

2.2 Ad Hoc Routing

To gain more insight into the inner workings of MANETs, we will now look into how packets can be routed in ad hoc networks. In infrastructure-based networks, routing is performed by the AP (usually a router). Ad hoc networks do not have APs and so the nodes have to rely on each other for routing. The problem with this is that, as stated in Section 2.1.1, ad hoc networks do not have any intrinsic capability for multi-hop communication, which is imperative when establishing routes between nodes. Furthermore, if the nodes are mobile (i.e. MANET), routing becomes much harder

because the topology will constantly be changing, which will cause route failures [41, p. 63].

Thus, a routing protocol for MANETs should provide the network with multi-hop communication and dynamically adapt to topology changes. All this must be done efficiently owing to the limited resources available (see Section 2.1.2). Protocols for wired networks would not work well because they usually have high message overheads, which is not as big a concern for networks with fixed topologies and steadily powered nodes as it can be in MANETs. This led to many protocols being specially created for routing in MANETs [7, p. 179]. They can generally be categorized as either pro-active or reactive protocols. The former category is mostly composed of traditional wired routing protocols that were changed in order to better serve routing needs in dynamic topologies. The latter is based on a different paradigm for routing.

Bellow is a basic description of each category.

- **Pro-active Routing**: routing protocols for wired, infrastructure-based networks are pro-active. Some of these protocols can be optimized to work in MANETs. In this kind of routing, every node keeps a route to each of the other nodes in the network, regardless of whether it is going to be used. Some examples of pro-active routing protocols are: Destination-Sequenced Distance-Vector (DSDV) [50], Wireless Routing Protocol (WRP) [42], Optimized Link State Routing (OLSR) [13] and Topology Broadcast Based on Reverse-Path Forwarding (TBRPF) [45].
- **Reactive Routing**: nodes have no prior knowledge of the topology and so, every time a node needs to communicate with another, it will attempt to create a route to that node. This represents an on-demand approach to routing because nodes only become aware of routes when they actually need them. Some examples of pro-active routing protocols are: Dynamic Source Routing (DSR) [31], Ad hoc On-demand Distance Vector (AODV) [49] and Temporally Ordered Routing Algorithm (TORA) [48].

Pro-active routing is preferable in networks with low mobility nodes where communication sessions are short and sporadic [7, p. 179], because despite the initial overhead of learning all the routes, nodes can simply use a route and do not have to wait for it to be established when they need it.

Conversely, reactive routing is preferred on networks with high mobility nodes because it adapts quickly to topology changes (reducing the occurrence of route failures). It is also suited for situations in which a node will only communicate with a subset of nodes at any given time [7, p. 179]. Moreover, establishing routes on-demand is more appropriate when communication sessions are few and long,

because the overhead of route discovery will not happen too often and when it does happen, it will have been insignificant in relation to the communication messages exchanged.

A comparison of pro-active and reactive routing is summarized in tables 2.1 and 2.2.

Table 2.1: Circumstances in which reactive or pro-active routing is more appropriate

	High	Low
Communication frequency	Pro-active	Reactive
Communication session duration	Reactive	Pro-active
Node mobility	Reactive	Pro-active

Table 2.2: Pro-active versus reactive routing

	Pro-active	Reactive
Overhead of periodical route updates	High	None
Route discovery delay when starting communication session	None	High

2.3 Naming System

The fundamental purpose of a network is to provide its nodes with the capability of communicating with each other. A source node should be able to simply point to the destination node in order to contact it. Any given node must know how to select the node with which it wants to communicate from the set of nodes that compose the network. For this to be possible, an inescapable requirement has to be met: every node in a network must be uniquely identifiable.

An obvious way to distinguish between nodes is to use their physical or network layer addresses, but these are several bytes long and have no discernible semantic meaning, which is why they are hard to remember and therefore are not convenient for frequent use. A more user-friendly way of designating nodes is thus necessary, i.e., a naming system.

A naming system can basically be described as a service that translates names into Internet Protocol (IP) addresses (name resolution). Nodes are given human-readable names which are bound to their respective addresses. Mappings between names and addresses are kept in a server which can be queried. Thus, when node A needs to communicate with node B, A only needs to know B's name, because it can ask the naming server which address corresponds to that name.

2.3.1 The Domain Name System (DNS)

Because the Internet is a global network, it cannot not rely on a single naming server to resolve names. Instead, a distributed approach had to be conceived. DNS is what we call the naming system used in the Internet. It is a globally distributed hierarchical naming system that resolves domain names into IP addresses. The DNS hierarchy is structured as a tree-like namespace that is partitioned into subsets of nodes (i.e., zones). Nodes in the same zone are given domain names according to the subdomains that compose a path from themselves to the root of the tree and the responsibility of resolving their names is given to a specific name server, which is the authoritative name server for that zone.

The most important concept of the DNS — and on which all of its components are based — is its reliance on a hierarchy. It is not overly complicated to build a hierarchy tree of name servers when you have a stable topology. The problem arises when we try to use DNS as a naming system for a network that has a dynamic topology. Because nodes can leave the network at any moment, it becomes difficult to entrust them with the task of resolving names. Name servers should be fixed nodes that are always reachable, which is clearly not the case in a MANET. DNS is therefore not a suitable candidate to be used as a naming system in MANETs.

2.3.2 Name Resolution in MANETs

As we explained in Section 2.3, a naming system is a fundamental part of any network. This becomes even more apparent when dealing with MANETs because the same node can join and leave the network multiple times, which can result in a different IP address for that node each time it joins the network. Thus, relying on fixed higher layer names for nodes is a necessity.

However, creating an effective and efficient naming system for MANETs is no easy task, many proposals have been made (see Chapter 3) but none of them have been standardized. This is most likely due to the fact that most of the decisions that must be made when conceiving a protocol for a MANET, heavily depend on the characteristics of the network that protocol is meant to be used on, i.e., level of mobility, amount of nodes, communication frequency, etc. These divergences make it difficult to build a single unifying naming system that could work equally well on different kinds of MANETs. It is therefore crucial that we look into the most important decision one faces when conceiving a MANET naming system: the architecture.

2.3.2.1 Architecture for a Naming System

Different MANETs, call for very different architectural approaches. When it comes to choosing the right architecture for a MANET naming system, the most determinant aspect is the size of the network, which means that the type of architecture heavily depends on the ultimate purpose of the MANET. An architecture that is efficient in a small MANET might not be as efficient in a larger one (and vice-versa).

Typically, there are three different approaches [7, p. 181] to name resolution in MANETs: totally distributed, centralized and partially distributed architectures. We will now look into some generic models that show how name resolution and its associated tasks are performed for each of these architectures, after which we will make a comparison between them pointing out the situations in which they would be more beneficial while identifying some of their advantages and disadvantages.

Note: borrwoing the terms User Entity (UE) and Named Entity (NE) from [7, p. 181], we will use them to refer to a node requesting a name resolution and to the node whose name is being resolved.

- **Totally Distributed Architecture**

  To resolve a name in DNS, we query a node that was specially created and configured for that function, i.e., the Name Server (NS). In a totally distributed architecture, there are no special nodes in charge of resolving names. Every node in the network is equal and tasked with exactly the same functions.

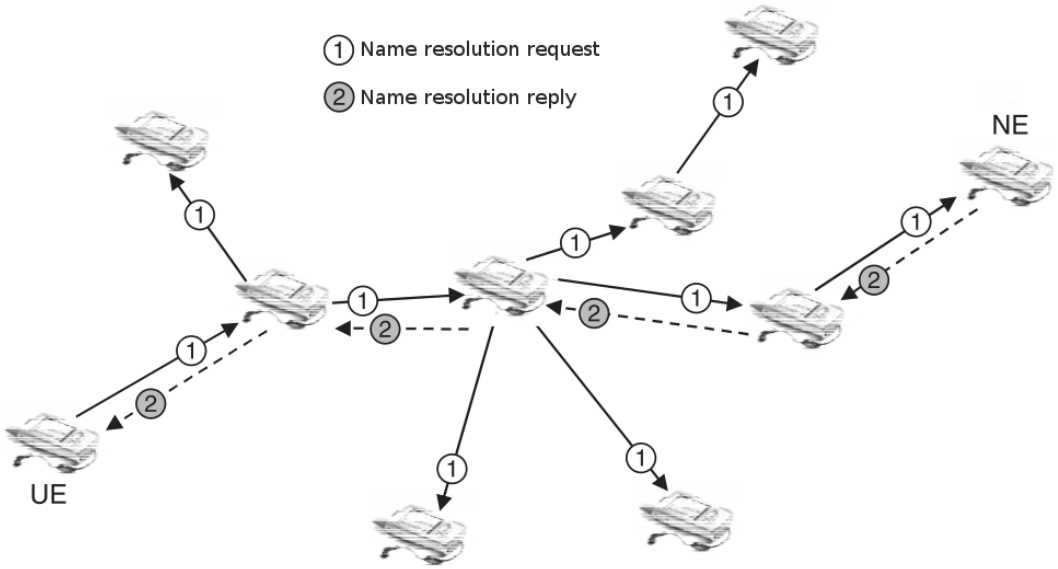

Figure 2.2: Name resolution process in a totally distributed architecture [7, p. 182]

To find out the address of an NE, a UE floods a Name Resolution Request (NREQ) message throughout the network. Every node that receives that message, responds with a unicast Name Resolution Reply (NREP) only if the name whose resolution is being requested is its own name, otherwise it just continues to flood the request. This exchange of NREQs and NREPs is illustrated in Figure 2.2.

Optionally, caching of name-to-address bindings could be done by nodes upon receiving an NREP [7, p. 183]. This would mean that an NREQ might not have not travel all way to the NE, because an intermediate node might have the requested resolution in its cache and would immediately respond with an NREP.

- **Centralized Architecture**

  In a centralized architecture, one of the nodes in the network is elected to be the NS. This node will periodically flood messages announcing its presence. Upon receiving an NS announcement, nodes will register their names using the address of the NS they just leaned about.

  The server caches all the names in the network and can therefore receive unicast NREQs from other nodes, to which it will respond with a unicast NREP to the requesting node. All of these processes can be seen on Figure 2.3.

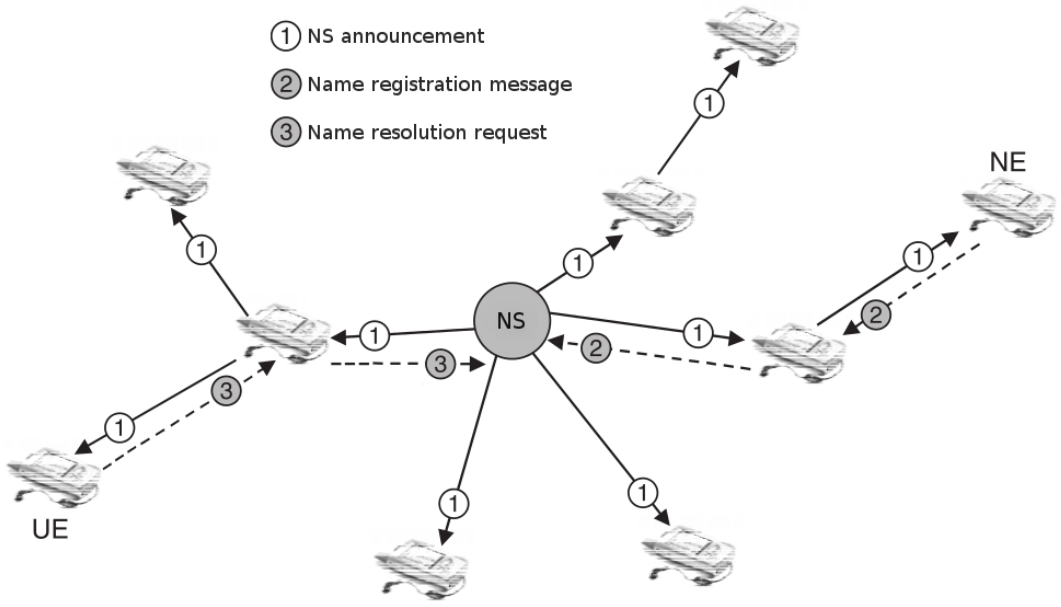

Figure 2.3: NS announcement flooding, name registration and name resolution request in a centralized architecture [7, p. 183]

- **Partially Distributed Architecture**

  For larger MANETs, it might not be efficient to have only one NS responding to all NREQs. This architecture is a modification of the centralized approach that enables a MANET to have multiple NSs instead of just one. The bindings of every node will thus be spread across the network in various servers.

  Each NS periodically floods announcement messages to the network (the scope of the flooding can be constrained to a certain number of hops [7, p. 182], this way, nodes will only know the address of the NS closest to them). Every node will then register its name with their NS and send unicast NREQs to that same NS.

  Having multiple NSs requires some kind of synchronization mechanism between them so that a UE can resolve a name of an NE that is not under the scope of the same NS (see below).

2.3.2.2 Requirements and Characteristics of a Naming System

Now that we have gone over the general architectures for a MANET naming system, we are now prepared to establish what should be expected from it. To be a suitable candidate for name resolution in a MANET, there are certain tasks a naming system should be able to perform [40, p. 3]. Below is a list of requirements[1] and characteristics that we consider a MANET naming system should meet. It is important to note that all of these should be implemented in a way that takes into account the issues discussed in Section 2.1.2 and thus should be resistant to topology changes and reduce the message overhead to a minimum because of the limited resources typically available to mobile devices. Ways in which these requirements could actually be implemented, are presented in Chapter 3.

These are requirements/characteristics that apply to any of the aforementioned architectures.

- **Namespace management**
  - **flat or hierarchical namespace:** names should be totally unstructured (flat) or reflect logical organizational affiliation [23, p. 1].
  - **naming convention:** the name of a node should be chosen by the user or by the protocol.
  - **name conflict detection/resolution:** if the protocol is charged with the task of choosing the names of the nodes, then that will probably reduce the occurrence of name conflicts, thereby reducing the overhead of name conflict resolution. On the other hand, those

[1] The general categories of requirements follow [40, p. 3] with some changes and additions we considered appropriate.

names would not be user-friendly. If the responsibility falls upon the user to choose its own name, then it would probably be more user-friendly, although that would result in a higher probability of name collisions [44, p. 6].

An alternative would be to combine both approaches. The user chooses part of the name so it can be user-friendly and the name system chooses the other part to guarantee uniqueness. If we cannot guarantee uniqueness, then there should be some mechanism for detection and resolution of said conflicts. However, it is desirable to have such a mechanism anyway, because even if names are unique within the same MANET, conflicts can still happen due to network mergence (see Section 2.1.2).

– **name registration:** after the name of a node has been chosen (be it by the protocol or by the user), there should be some way to register that name in the network so that other nodes become aware of their presence.

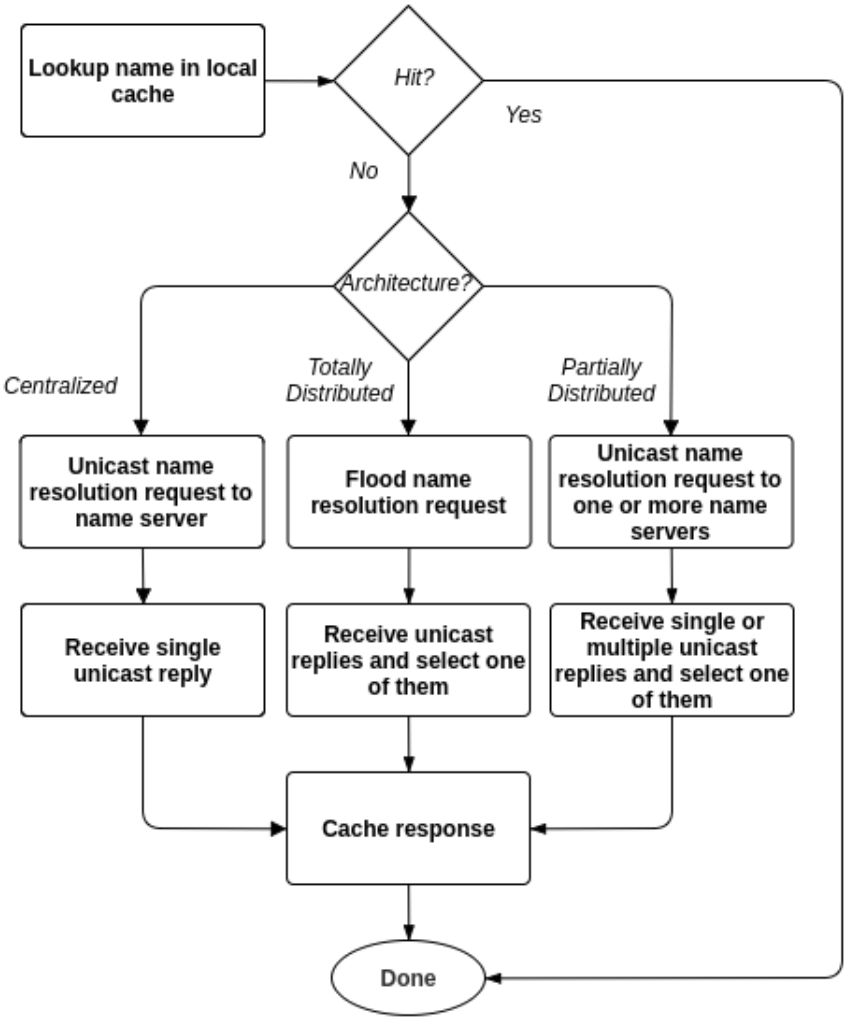

Figure 2.4: Steps for name resolution depending on the architecture chosen for the naming system

- **Name resolution**
  - **Name resolution request/reply distribution:** every node must be able to request a name resolution from the network and receive the respective reply. The steps a node can take in order to resolve a name are illustrated in Figure 2.4.
  - **Database lookup:** nodes tasked with maintaining name-to-address mappings should have a local lookup functionality so they can respond to name resolution requests.
  - **Cache management:** intermediate nodes through which a reply is routed should cache the name-to-address mapping in that reply if they do not have it already. Furthermore, cached entries should expire after some time has elapsed (TTL), to ensure that replies are fresh. Ideally, the TTL should be defined by the node to which the name belongs.
  - **Compatibility with DNS:** if a node in a MANET is connected to the Internet, then it should be possible for nodes to resolve outside names with DNS using that node as a gateway[2] [17].
- **Security**
  - **Verification of message authenticity/integrity:** there should be a way for nodes to make sure that a reply to a name resolution request came from a trusted source and was not altered.
  - **Prevention of Denial of Service (DoS) attacks**

As said before, these tasks must be performed regardless of the architecture that is being used. If the architecture is not a totally distributed one (i.e., it is centralized or partially distributed), then a series of additional challenges is introduced:

- **Name server election:** some mechanism for electing which node(s) will act as NS(s) is required. An ideal mechanism would elect nodes that are positioned at the center of densely populated areas [23, p. 2].

  Due to the mobility of nodes, NS election must occur periodically because the topology might have changed enough that the previously elected NSs are no longer appropriate. Election must also be performed whenever an NS is disconnected from the network and when either network partitioning or mergence happens [40, p. 13]. This also implies that there would have to be a backup of the names registered in the old NS so they would not be lost upon re-election.

[2] A MANET node connected to the Internet shall henceforth be referred to as a *gateway*.

- **Name server absence detection:** any node in a MANET might be disconnected at any given time. This becomes a bigger problem if that node also happens to be an NS. In order to run the election algorithm when some NS is disconnected, the network must have some mechanism that allows it to detect such an event [40, p. 12].
- **Name server discovery:** when a node joins the MANET, it must have some way of discovering the address of a name server. The node can pro-actively ask the network for the address of an NS (pull method) or passively wait to receive some message indicating that address (push method). Because there might be multiple NSs, a joining node must have some way of deciding which server it is going to use.
- **Name server synchronization:** a MANET with a partially distributed naming system can have multiple NSs. In the interest of consistency, they should all have the same cached bindings. Furthermore, if an NS is in charge of a certain group of nodes and it receives an NREQ for a node that is managed by another NS, then the first NS must know how to respond, which means that some synchronization system between NSs must be in place [40, p. 13].

One important aspect to keep in mind is that, all of these tasks should be performed in a manner that is transparent to the application layer. We should not expect applications to adapt to a new naming system, they should function as if they were in a regular infrastructure-based network using DNS, without perceiving all that is being done underneath.

2.3.2.3 Choosing the Right Architecture

Now that we have established the tasks a MANET naming system must be able to perform, we should now be able to determine which of the architectures mentioned in Section 2.3.2.1 will have a better chance of performing the required tasks efficiently, depending on the characteristics of the MANET for which we are conceiving a naming system.

Due to the lack of infrastructure and adaptability to frequent topology changes, a totally distributed architecture is perhaps the most natural fit for a MANET where nodes are highly mobile. The constant topology changes make it difficult to choose any one node to perform some special task, in this case, to be the name server. Furthermore, totally distributed architectures have no need for NS election or synchronization mechanisms. However, the main issue with a totally distributed architecture is its heavy reliance on flooding techniques. Although flooding is simple and easy to implement, its performance is poor, because it generates a large amount of messages. Consequently, if the MANET

grows to a considerable size, the message overhead can become excessive and overwhelm the network. A totally distributed approach might therefore not be the best option for MANETs in which scalability is needed. Moreover, if caching on intermediate nodes is used, then an additional caching storage overhead is imposed on the network.

Another issue is that, without a central managing entity, it becomes harder to detect and resolve name conflicts. Flooding would inevitably have to be used, imposing yet more message overhead.

If we can, however, guarantee a relatively stable topology, we might consider using special nodes to perform NS functionalities. A centralized architecture eliminates the overhead of flooding messages by using only unicast messages to and from the NS. Every node in the network will request name resolutions from the same server, because caching of bindings is not performed by intermediate nodes, which causes less overall storage overhead. But as there is only one NS, if the number of nodes rises, then the performance of the network will decline. Thus, apart from being a single point of failure, having a single server for a large MANET is infeasible, which means that a centralized approach is only suitable for smaller MANETs because it does not solve the scalability problem. Furthermore, it incurs on the additional overhead of NS election.

Table 2.3: Summary of advantages and disadvantages of each naming system architecture

	Totally Distributed	Centralized	Partially Distributed
Advantages	– No single point of failure – No need for election mechanism – No need for NS synchronization	– Easier to guarantee unique names – Caching only in NS (less overall storage overhead) – No need for synchronization	– Easier to guarantee unique names – Caching only in NSs (less overall storage overhead) – No single point of failure – Higher performance – Higher scalability
Disadvantages	– Harder to guarantee unique names – Caching probably needed (caching storage overhead) – Low performance – Low scalability	– Needs election mechanism – Single point of failure – Low performance – Low scalability	- Needs election mechanism – Needs synchronization of NSs

If we are dealing with a sizeable MANET, then we can extend the centralized architecture to have, not one, but multiple NSs, which is what we call a partially distributed architecture. This approach also avoids flooding by unicasting NREQs and NREPs, but contrary to the centralized version, it does provide higher performance and scalability due to distributing the NS tasks across multiple conveniently positioned nodes (and thus also eliminating the single point of failure problem). However, in addition to the NSs election mechanism, partially distributed architectures also have to concern themselves with the NSs synchronization. As the MANET grows, so does the complexity of both the election and synchronization mechanisms.

For an easier reference, a summary of the general advantages and disadvantages of the naming system architectures that were examined, can be seen in Table 2.3.

In light of the above, it is clear that deciding on an architecture for a MANET naming system, is no straightforward task. It might, perhaps, be impossible to conceive a protocol that would work equally well for small and large MANETs, for example. There are many factors to be taken into account and some of them might be out of our control (like node mobility or number of nodes). The unavoidable conclusion is that the characteristics of our protocol will depend on the characteristics of the MANET and the way in which it is going to be used.

Chapter 3

Related Work

Having been through the concepts presented in Chapter 2, we are now aware of what a Mobile Ad Hoc Network (MANET) is, how it behaves and how some of its appealing characteristics may, at some point, also become limiting factors. We have also established what tasks a MANET naming system should be able to perform. But before we venture into developing a new naming system for MANETs, we must first study what has already been done in this field, in order to assess if we overlooked something in the previous chapter, as well as learn from mistakes that might have been made by other researchers and thus, avoid the consequent pitfalls.
To that end, we gathered the eight most relevant proposals (described in eleven different articles) we were able to find[1], that have been published over the years and evaluated them according to the requirements established in 2.3.2.2. We end this chapter with a comparison of between all proposals.

3.1 Classification of Studied Proposals

The most distinctive property of the proposed naming systems is, by far, the architecture. Most of the other characteristics can only be defined after this one, because it is crucial to know if there will be any nodes with specially assigned functions. This led us to classify all the proposals according to their architectures. Then, as per the list of required tasks defined in Section 2.3.2.2, we will go over the ways in which these tasks are performed in the different proposals.

Note: in cases where none of the proposals in question mention how to perform a certain task, that task will simply be omitted. For example, Ahn [2] does not mention any kind of security in his

[1] Some of these proposals are described in an extensive survey conducted by Masdari et al. [40] and other relevant ones (that either cited or were cited by Masdari) were included as well.

centralized architecture proposal. Security related tasks are therefore omitted from Section 3.1.1.

3.1.1 Centralized

Recall from Section 2.3.2.1, that in a centralized architecture, there is only one node that is elected to act as the Name Server (NS) for the entire MANET. To allow for scalability, this is probably not the best solution (see Section 2.3.2.3), which is why from the set of all selected proposals, only one of them has a centralized approach to name resolution. Notwithstanding, we chose to include it here for the sake of completeness. The proposal is a naming system devised by Ahn [2], called *Manet DNS*. The following is a description of the manners in which *Manet DNS* performs the tasks established in Section 2.3.2.2.

- **Namespace management:** each node must choose its own flat name and register it by sending a Domain Name System (DNS) register message to the NS. Because all the names in the MANET are stored at the same NS, it can tell new joining nodes that try to register names that are already taken, to choose a new name. This means that name conflicts can only happen upon network mergence. However, this possibility is not mentioned by the author.

- **Name resolution:** to resolve a name, nodes must send a unicast Name Resolution Request (NREQ) to the NS, which retrieves the binding from its cache and then responds with a unicast Name Resolution Reply (NREP). To keep the NS cache up to date, nodes must notify the server before leaving or, if a node is unexpectedly disconnected due to its mobility, other nodes that try — and fail — to communicate with it, will warn the NS. After a certain number of warnings, the NS deletes the respective entry from its cache.

- **Name server discovery, election and absence detection:** when a node joins the network, it broadcasts a DNS server solicitation message in order to discover the address of the NS. The NS (or any intermediate node that already knows the NS' address) responds with a unicast DNS server advertisement[2]. If there is no response to the solicitation message, it means that there is no NS available, and so the node broadcasts a DNS server advertisement announcing itself as the new name server (this is how the NS is elected — it elects itself).
  However, there is a different way for an NS to get re-elected. When two MANETs merge, there will probably be two NSs present in the newly formed network. This is detected when one of the NSs receives an advertisement from the other. They go through a tie breaking procedure

[2] In addition to being sent as responses for server solicitations, server advertisements are also sent periodically

(vaguely described as being based on the servers' addresses [2, p. 3]), and the NS determined to be have the lowest priority, is forced to handover its cache and stop acting as an NS.

3.1.2 Totally Distributed

As explained in Section 2.3.2.1, in a totally distributed architecture there are no NSs, which means that all nodes must respond to NREQ messages and they must collaborate to perform tasks such as name conflict resolution.

Because there is no central entity to manage communication, nodes will be reduced to using flooding techniques to send NREQs. For large MANETs, this is not feasible due to the fact that the amount of generated messages might overwhelm the network and consequently, affect performance. Despite their lack of scalability, most of the studied proposals take the distributed approach to a MANET naming system.

3.1.2.1 Multicast-based

In multicast communication, nodes can belong to groups by joining certain multicast addresses that are associated with said groups. Multicast trees composed of nodes that are members of the same group are then formed. When a message is sent to a multicast address, only the nodes belonging to the associated group will receive it.

The only totally distributed proposals whose operation is based on multicast communication, were published by Gottlieb [20] and Jeong [28–30]. They are also the only ones that choose to employ a hierarchical namespace. This is due to multicast being the easier way of communicating with groups of nodes established by a hierarchy of subdomains.

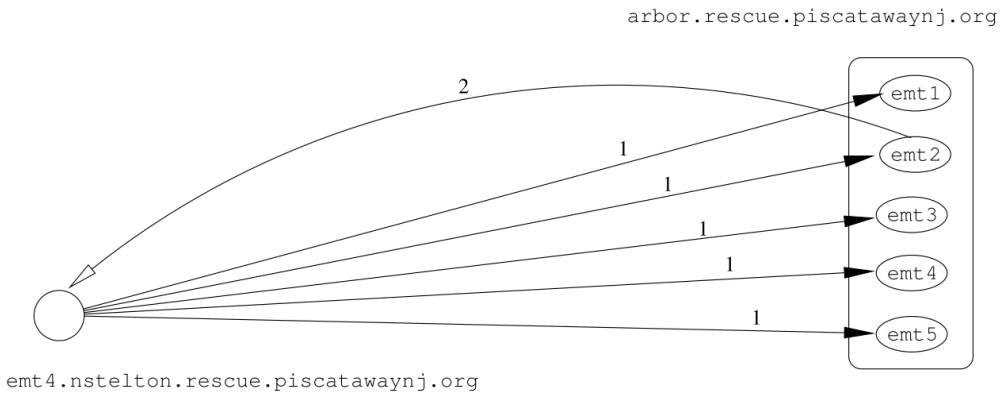

Figure 3.1: Name resolution process in MOSS [20, p. 3]

In Gottlieb's proposal [20] — named MOSS — every node is pre-configured with (1) a list of all the multicast group names and respective multicast addresses that will be present in the MANET, and (2) a domain name such as *emt1.arbor.rescue.piscatawaynj.org* that reflects the multicast groups to which that node will belong. In this case, node *emt1* belongs to a group named *arbor*, for example. This means that there is no need for name registration when the MANET is initiated, a node simply has to check which multicast groups it belongs to (based on its own name), and then join those groups.

A multicast group can be considered as authoritative for names in a certain subdomain. Therefore, to resolve a name in MOSS, node A must check its list to find the multicast address associated with the domain of node B. Node A then sends a standard DNS query to that address. When node B receives the query, it unicasts a reply back to node A. This process is illustrated in Figure 3.1, where node *emt4.nstelton.rescue.piscatawaynj.org* tries to resolve the name *emt2.arbor.rescue.piscatawaynj.org*. Because names are chosen prior to deployment, every node in every group will have a unique name and therefore, name conflicts will not occur (not even upon MANET mergence).

The other multicast-based approach called Ad Hoc Name Service System (ANS), was proposed by Jeong in three different articles [28–30]. We will mostly focus on the last one [30], as it is the culmination of the first two. ANS requires the shared tree based reactive routing protocol Multicast Ad Hoc On-demand Distance Vector (MAODV) [6] in order to form multicast groups. Each group has a Group Leader which periodically announces itself to the whole network, indicating the existence of its multicast group. Thus, nodes will receive announcements from multiple Group Leaders and can then choose what groups to join.

These groups, however, are not related to name resolution, because apart from them, there is also a general multicast group which all nodes should join. It is this group's address that will be used for name resolution. Each node runs an ANS Responder and an ANS Resolver. Nodes must register their names (of the type *node1.manet*) with their own responders. A node's resolver multicasts a DNS query request message and each ANS Responder with the binding of the requested domain name, unicasts a DNS query response message back to the ANS Resolver of the requesting node. Whenever a node receives a name resolution reply, it caches it and sets a timer that, when expired, will cause the node's responder to invalidate the respective binding, thus assuring that cached bindings are never too stale.

If some of the responses received by a node — after having sent a resolution request — are different addresses for the same name, then there is some name conflict that must be resolved. The

requesting node sends warnings to all the nodes that responded — except for the first one — asking them to change their names.

When it comes to resolving Internet names using DNS through a gateway (see Section 2.3.2.2), Gottlieb's MOSS [20] does not consider it at all (despite using the standard DNS message format for NREQs and NREPs). Jeong's ANS [29, 30] on the other hand, defines that a gateway must notify the nodes in the MANET of its address, so that they can send recursive DNS queries to that gateway to resolve global DNS names.

Before moving on to non-multicast-based approaches, we should acknowledge two multicast name resolution schemes that are used to resolve names in small networks where no name server is present: Apple's Multicast DNS (mDNS) [12] and Microsoft's Link-Local Multicast Name Resolution (LLMNR) [1]. Both LLMNR's and mDNS' queries are only for local communication on a particular physical network segment, they do not propagate beyond the link-local scope, which means they are not suitable for multi-hop MANETs.

3.1.2.2 Integrated with Lower Layer Protocols

Another type of totally distributed naming system, was proposed by Engelstad [17, 18] and Hu [24]. They make use of the underlying reactive routing protocol to perform name resolution, thus integrating name resolution into the route discovery process. The same idea is used by Jelger [27], although he goes even deeper and also combines these with link-layer address resolution. Although all three mentioned proposals involve integrating name resolution into lower layer protocols, Jelger's approach is quite different from the other two, so we will describe them first.

The second Englestad paper [17] expands on the first [18], so we will rely on the second one. Both Englestad's [17] and Hu's [24] proposals are very similar and follow basically the same idea, which is why the description bellow works for both of them, although there are some differences that set them apart, which will be mentioned as needed.

This idea requires the use of a reactive routing protocol — that only needs one flooded broadcast message (plus the respective unicast response) to discover a certain route [7, p. 180]. Engelstad [17] claims that the naming protocol must be optimized in respect to the routing protocol and so, if name resolution could be performed simultaneously with route discovery, no flooding overhead of name resolution would be added. Figure 3.2 a), shows that as many as three flooded broadcasts might be needed to resolve a name, if route discovery and name resolution are performed separately:

- Step 1: the User Entity (UE) broadcasts an NREQ;
- Step 2: the Named Entity (NE) receives the NREQ, but to respond, it needs to discover a route to the UE and so, the NE broadcasts a Route Request (RREQ);
- Step 3: once the UE has learned the address of the NE, it must discover a route to it and so, the UE broadcasts yet another RREQ.

This led Engelstad [17] and Hu [24] to the realization that there must be an effort to reduce flooded broadcasts to a minimum. To that end, we could combine route discovery with name resolution, which would reduce the number of broadcasts to just one [7, p. 184]. Name resolution would then be performed as illustrated in Figure 3.2 b):

- Step 1: the UE inserts an NREQ inside of an RREQ message (piggybacking) and broadcasts it;
- Step 2: as part of the previous flooding, when the NE receives the message, a return route back to the UE is already formed. The NE piggybacks the NREP on the back of a Route Reply (RREP) and unicasts it along the return route;
- Step 3: from now on, the UE will be able to communicate with the NE by unicasting messages directly to it.

Hu [24] improves on Engelstad [17] by introducing one property, NAME, into the routing table of each node. This way, intermediate nodes can also respond to NREQs, providing the requester with the name binding and route to the destination node.

To provide transparency to the application layer, the routing module must be modified to catch standard DNS queries generated by the nodes' DNS resolvers and translate them into RREQs with piggybacked NREQs (see Figure 3.3), which are then flooded though the MANET, as described above.

Only Engelstad [17] elaborates on the fact that a MANET naming system should interoperate with DNS so that it can resolve both local names and names from the DNS through a node acting as a DNS proxy (this is what we defined earlier as the *gateway*). As shown in Figure 3.4, this is again done by NREQ/NREP piggybacking. The gateway receives the request, translates it into a

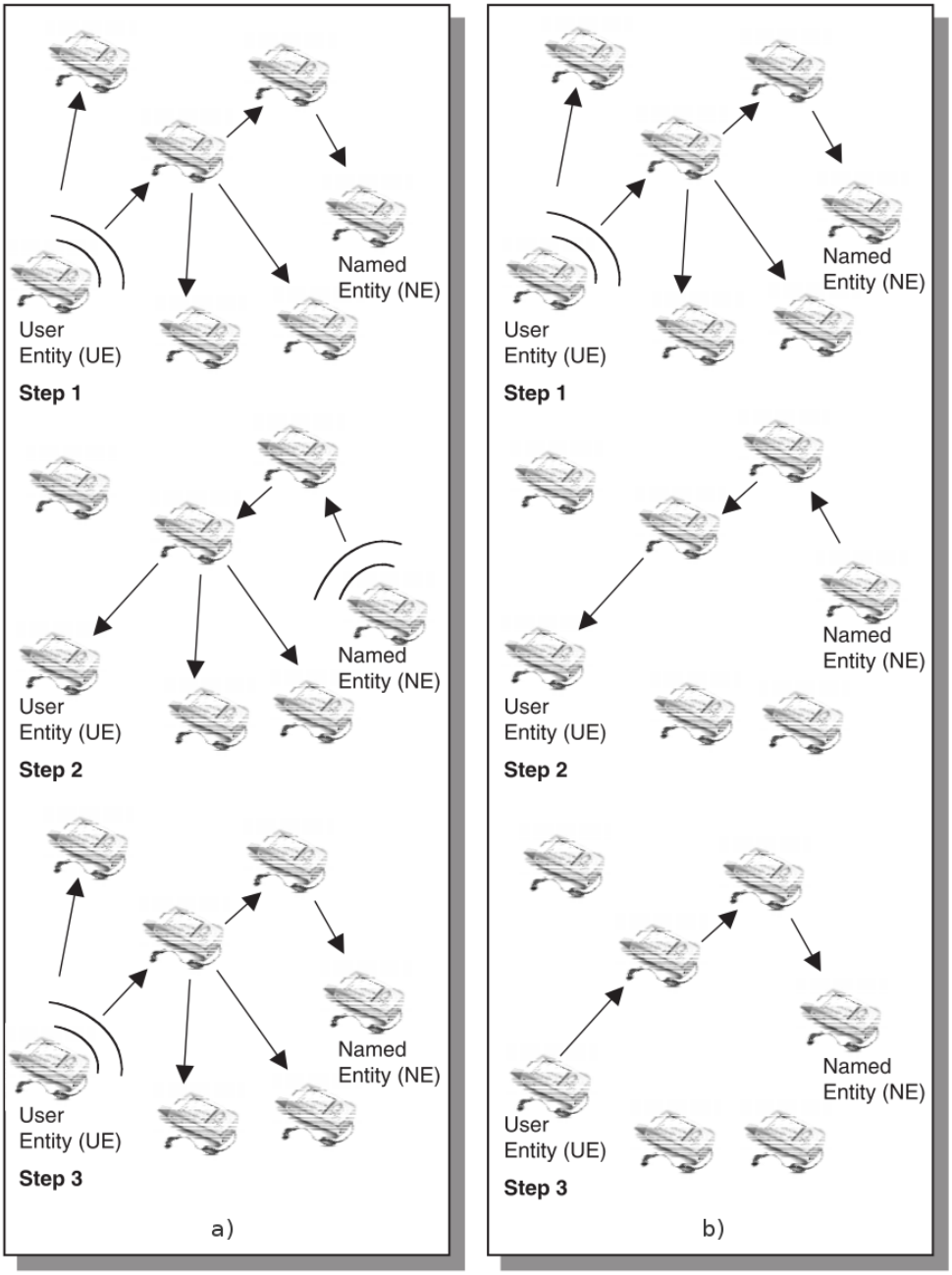

Figure 3.2: Number of broadcasts needed to perform name resolution with a) separate route discovery and name resolution and b) simultaneous route discovery and name resolution [7, p. 185]

standard DNS query and uses an external DNS server to resolve it. Once it gets the reply, the gateway translates it back to an RREP+NREP and unicasts it to the requesting node (this sets up a route for subsequent communication over the gateway).

A notable absence in both Engelstad [17] and Hu [24], is the lack of a name conflict detection/resolution mechanism. Furthermore, they do not mention if the namespace is flat or hierarchical or who

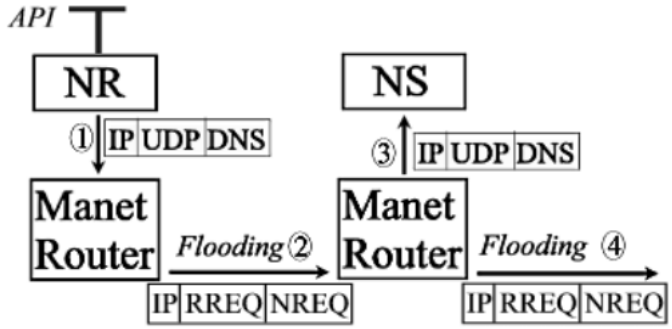

Figure 3.3: Routing module translates DNS query into RREQ+NREQ before flooding it [17, p. 3]

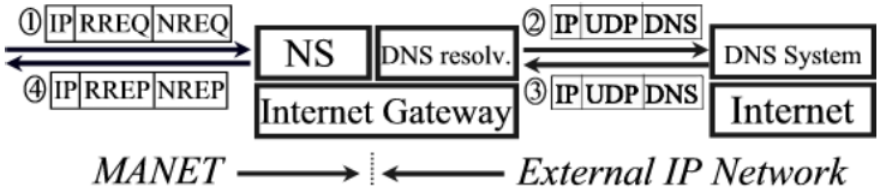

Figure 3.4: Name resolution through MANET gateway [17, p. 5]

chooses the names of the nodes.

Recall that, apart from Hu [24] and Engelstad [17], there was another totally distributed naming system that was integrated with lower layer protocols. It was proposed by Jelger [27] and is called Lightweight Underlay Network Ad hoc Routing next generation (LUNARng). It combines name resolution, route discovery and link layer address resolution into a single operation.

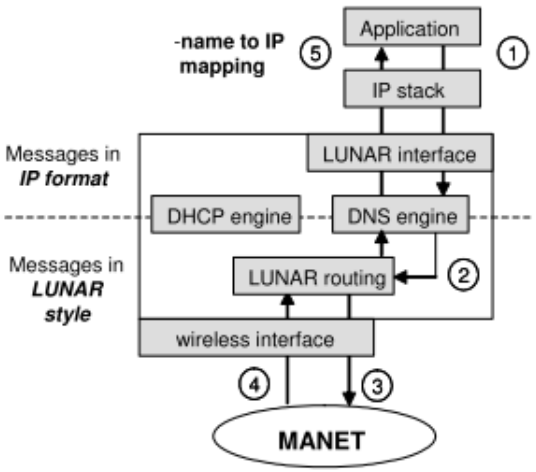

Figure 3.5: Name resolution in LUNARng [27, p. 3]

LUNARng is based on the Lightweight Underlay Network Ad hoc Routing (LUNAR) [57] reactive

routing protocol, that creates a virtual network interface that sits between the network and link layers (see Figure 3.5). This way, the LUNAR module can intercept all the traffic to and from he node. When a node joins a MANET, it will receive — from the virtual Dynamic Host Configuration Protocol (DHCP) server in the LUNAR module — an Internet Protocol (IP) address and the address of a the virtual DNS server that is also part of the module, so that it can intercept DNS queries. It also intercepts link layer address resolution messages from Address Resolution Protocol (ARP) and Neighbor Discovery (ND) messages for IPv4 and IPv6, respectively.

Name resolution in LUNARng (see Figure 3.5), is performed in a way similar to Engelstad's [17] and Hu's [24] proposals. It is also done simultaneously with route discovery and also translates standard DNS to special format messages before sending them to the MANET. The difference is, when a node receives a response to an RREQ+NREQ, that response also includes the link layer address of the next hop towards the node whose name was just resolved.

LUNARng specifies that, whenever an application requests a resolution for a flat name, the virtual domain *net.lunar* is appended to that name by the operating system[3]. Otherwise, if the name is a Fully Qualified Domain Name (FQDN), nothing is appended. Then the LUNAR module will perform the resolution described above if the name is under the *net.lunar* domain or it will forward the request to an Internet gateway, if one exists, so normal DNS resolution can be performed.

Lastly, it's worth noting that, as Engelstad [17] and Hu [24], Jelger also does not propose a name conflict detection/resolution mechanism.

3.1.3 Partially Distributed

Recall from Section 2.3.2.1, that in a partially distributed MANET naming system, there are multiple nodes that are elected to act as NSs for the network. From the set of proposals that we studied, there are two that choose to adopt this approach. Namely, Hong [23] and Nazeeruddin et al. [44]. They sought to develop a protocol that would combine the robustness of totally distributed protocols and the efficiency of centralized protocols [44, p. 3]. Furthermore, they claim that, despite the overhead of NS electing and synchronization in a partially distributed architecture, name resolution in a MANET might benefit from some degree of redundancy on the NSs.

[3] It is not clear how the operating system will be told to do this

3.1.3.1 Integrated with Stateful Auto-configuration Protocol

Nazeeruddin's proposal [44], called MANET Naming Service (MNS), makes use of a stateful auto-configuration protocol [43] to compensate for the overhead of the election and synchronization on NSs. This protocol dynamically selects nodes — called Address Agent (AA)s — that will have a range of IP addresses that they will assign to other nodes. Each AA maintains a record of all the addresses that it has assigned to others and their respective link-layer addresses [44, p. 3]. Moreover, AAs will multicast messages to update each other regarding their database changes. If a certain AA's update is not received for some time, it is considered unreachable and a new election process is initiated.

The main idea in MNS, is to extend AAs to act as NSs as well. To that end, the AAs' databases are modified to include an extra *Name* column. As we can see in Figure 3.6, the MNS query handler receives both name and auto-configuration queries and then forwards them to the appropriate module.

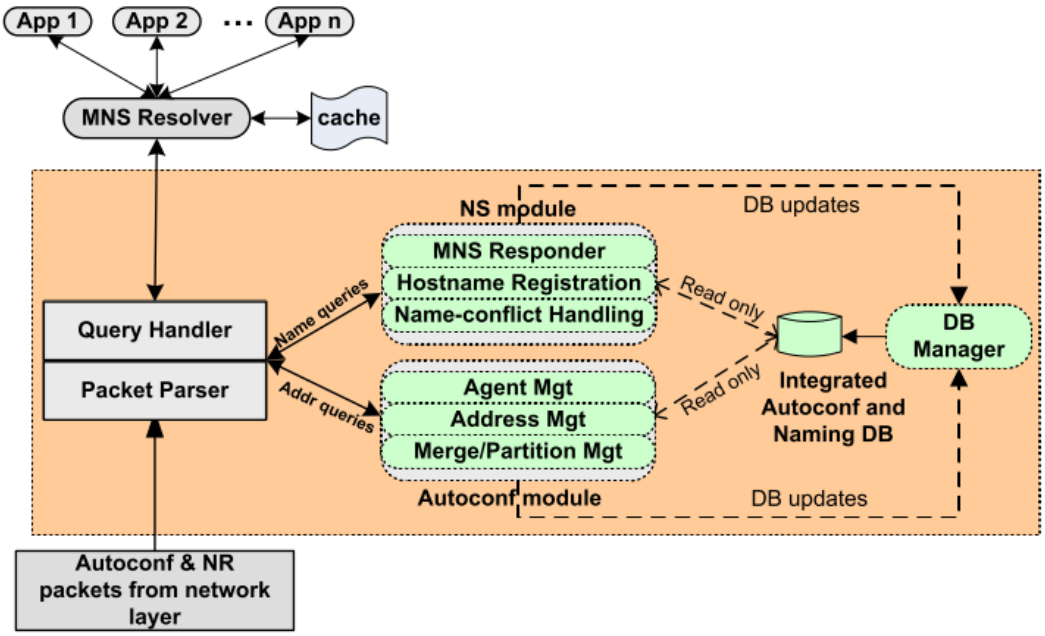

Figure 3.6: The MNS module architecture [44, p. 4]

MNS uses hierarchical names, although it allows nodes to use a *.manet* default domain that will be appended to the flat name chosen by the user. As part of the auto-configuration process, a node will register a name with an NS[4], which will check it for uniqueness. If there is a conflict, the NS will append the Host ID (HID) of the assigned IP address to that name. Name conflicts will also be detected on NS synchronization.

[4] Nazeeruddin does not specify how this NS is discovered by the joining node.

Another attribute received by a node during auto-configuration is a Partition ID (PID). This will serve to resolve name conflicts on network mergence. The smaller MANET will resolve the conflicts by appending the HID to the conflicting names.

Every node runs an MNS Resolver and NSs also run an MNS Responder. The responder receives NREQs from the local resolver or from resolvers from other nodes. To resolve names, nodes unicast NREQs to the NS responsible for their partition and the reply will also be sent in unicast. However, after some amount of failed attempts, the node will multicast the request. When an NS or the node (whose name resolution is being requested) receives the request, it responds with a unicast NREP. In other words, the system falls back into a fully distributed architecture when no NS is available.

Nazeeruddin's MNS [44] also considers the possibility of MANET gateways. When a resolver receives an NREQ with a local domain name, it resolves it locally. Otherwise, the resolver frames the query as a standard DNS query and forwards it to the gateway, which will resolve it using DNS.

3.1.3.2 Distributed Hash Table (DHT)-based

The other totally distributed scheme was proposed by Hong [23] and is called MANET Distributed Naming System (ADNS). Hong considers that redundancy is a desirable feature in a MANET naming system because it provides fault tolerance, making it a more robust protocol. To that end, Hong uses a DHT to spread name information throughout the nodes.

ADNS resorts to clustering algorithms [3, 19, 38, 61] to elect the nodes that will behave as NSs. These nodes will then announce their presence to the network. To reduce the overhead of the election process, clustering messages can be piggybacked on the messages of a proactive routing protocol.

Every node in the network maintains a table (the NS table) with all the NSs for which it has received advertisements. Eventually, all nodes will have received the advertisements from all the available NSs. Plus, the NS table is alphabetically ordered, which means that all nodes will have the same table. Furthermore, it is divided into as many segments as the level of redundancy desired.

In Figure 3.7, the level of redundancy is 2, which means that (1) each node will have two NSs that can resolve its name and (2) each segment on an NS table will contain information on two NSs (e.g., U's table has two segments, each with two NSs). When node U joins the MANET, it must choose its flat name and register it. To do so, U hashes its own name, using a pre-defined function, obtaining the number 0. This means that every NS at position 0 of each segment (namely, A and C),

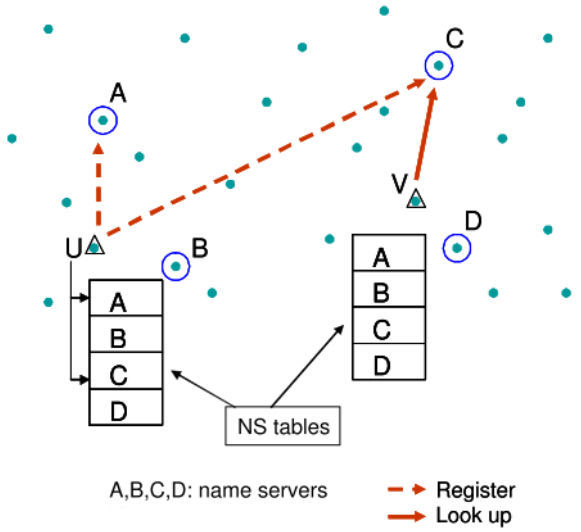

Figure 3.7: Name registration and resolution in ADNS [23, p. 3]

will be the NSs responsible for resolving *U*'s name and so, *U* registers its name with those servers. Then, when node *V* wants to resolve *U*'s name, it also hashes *U*'s name in order to discover its NSs. *V* then unicasts an NREQ to *C*, as it is closer[5].

To keep name-to-address bindings fresh, nodes periodically check if their NSs have changed and, if they have, the nodes will register their names with the new servers.

Hong also defines that NS database handover must be performed whenever a new election of NSs is performed (due to drastic topology changes or to network mergence).

3.2 Security

A notable aspect of nearly every proposal, is the lack a security system. In fact, only Gottlieb [20] and Jeong [30] demonstrate some kind of security concern. However, Gottlieb [20] only briefly mentions the use of a pre-defined key shared by all MANET nodes to encrypt communications. While Jeong [30] uses Internet Protocol Security (IPSec) Encapsulating Security Payload (ESP) with a null-transform to authenticate name resolution replies. This is achieved by also having a pre-shared key and by the use of Hash-based Message Authentication Code (HMAC).

[5] Hong [23] does not explain how proximity is determined.

3.3 Comparison

To end this chapter, we will now take an overall view of the proposals we presented earlier.

As far as the architecture of the naming system is concerned, recall from Table 2.3 that the biggest problem with the centralized approach was its lack of scalability, due to the fact that a single NS would not be sufficient to manage name resolution in large MANETs. The only centralized proposal was Ahn's [2], which did not solve this problem.

In totally distributed proposals, we examined multicast-based (Gottlieb [20] and Jeong [30]) and lower layer-integrated (Engelstad [17], Hu [24] and Jelger [27]) approaches. The problem with multicast — apart from the overhead of messages for establishing the multicast tree — is that it is not connection-oriented, which means that packets are not acknowledged. This can cause high rates of packet loss, making multicast-based approaches to MANET name resolution unreliable[6].

Conversely, the schemes that chose to take the totally distributed approach without resorting to multicast, use broadcast instead, which still suffers from the lack of packet acknowledgement and still produces a large amount of message overhead because many nodes will receive NREQs that are not relevant to them. To compensate for said overhead, these proposals try to integrate name resolution in routines of underlying protocols that would have to be performed anyway (e.g., route discovery). We should note, however, that neither of these three proposals present a solution to name collision, which is a significant problem in a totally distributed environment.

In respect to partially distributed proposals (Hong [23] and Nazeeruddin [44]), they either use an auto-configuration protocol or clustering algorithms to elect NSs. In a MANET where nodes are highly mobile, choosing the wrong auto-configuration protocol or clustering algorithm, could lead to frequent re-election of NSs, which might produce too much overhead for the network to handle. However, a partially distributed approach provides better scalability than any other architecture.

To summarize what we have discussed in this chapter, we organized the most significant aspects of every proposal in Table 3.1 and Table 3.2.

[6] This issue can be addressed by the use of protocols such as Pragmatic General Multicast (PGM) [55] (formerly Pretty Good Multicast), which introduces Negative Acknowledgement (NACK)s for lost packets.

Table 3.1: Summary of studied proposals — Part 1

PROPOSAL	ARCHITECTURE	NAME SPACE	NAME ASSIGNMENT	REQUEST	REPLY	ROUTING PROTOCOL	IPv4	IPv6
Ahn [2] (MANET DNS)	Centralized	NA	User	Unicast	Unicast	AODV used in simulation (but not required)	NA	NA
Engelstad [17]	Totally distributed	NA	NA	Broadcast	Unicast	Reactive only	✓	✓
Hu [24]	Totally distributed	NA	NA	Broadcast	Unicast	Reactive only	✓	✓
Gottlieb [20] (MOSS)	Totally distributed	Hierarchical	Partly User	Multicast	Unicast	NA	✓	✓
Jelger [27] (LUNAR)	Totally distributed	Flat (FQDNs resolution requests are routed to the GW, if one exists)	Partly User	Broadcast	Unicast	Reactive only	✓	✓
Jeong [30] (ANS)	Totally distributed	Hierarchical	Partly User	Multicast	Unicast	Reactive only (MAODV used)	✗	✓
Hong [23] (ADNS)	Partially distributed	Flat	User	Unicast	Unicast	Clustering algorithm msgs for NS election can piggyback on proactive routing msgs	NA	NA
Nazeeruddin [44] (MNS)	Partially distributed	Hierarchical	User or Protocol (on collision)	Unicast (multicast if no NS is available)	Unicast	NA	✓	✓

Table 3.2: Summary of studied proposals — Part 2

PROPOSAL	COMPATIBILITY WITH DNS	SECURITY	TESTING/ IMPLEMENTATION
Ahn [2] (MANET DNS)	NA	NA	Tested: vs mDNS
Engelstad [17]	Uses DNS msg format; Gateways translate local NREQs into DNS queries and translate the DNS replies back to local NREPs	NA	NA
Hu [24]	NA	NA	Tested: vs Engelstad
Gottlieb [20] (MOSS)	Uses DNS msg format; Resolver must be modified to not dismiss response with src addr different from dst addr of query	Briefly mentions shared key to encrypt communications	NA
Jelger [27] (LUNAR)	LUNAR intercepts regular DNS requests and routes them according to the structure of the name (flat or FQDN)	NA	Implemented
Jeong [30] (ANS)	Resolution of global DNS names through recursive queries to gateway	IPSec ESP	Implemented (implementation not publicly available and no info on performance is given)
Hong [23] (ADNS)	NA	NA	Tested: Showed better performance than flooding-based name resolution
Nazeeruddin [44] (MNS)	If a resolution request is for an FQDN name, the resolver frames the query as a standard DNS query and forwards it to the gateway	NA	Tested: vs ANS

Chapter 4

Proposed Name Resolution Mechanisms

Taking into consideration all the Mobile Ad Hoc Network (MANET)-related concepts and the naming system proposals studied so far, we are now in possession of the necessary data to make an informed decision about how to build our own naming system protocol for a network in an ad hoc environment. In this chapter, we will substantiate our choice of a fully distributed architecture and present two different proposals for name resolution in a MANET depending on the type of routing performed by the network. In Section 4.2, we suggest a new routing protocol-independent approach for pro-actively routed networks, based on disseminating the name throughout the network along with the routes by immediately requesting the name from a neighbor after receiving a new route from it.
For networks that use reactive routing, we suggest, in Section 4.3, an implementation of *the integrate name request in route discovery* concept defined in [17], using Ad hoc On-demand Distance Vector (AODV) as the routing protocol.

4.1 Architecture

As stated in Section 2.3.2.1, the most crucial decision when conceiving a protocol for a MANET naming system, is the type of architecture to be used. This is due to the fact that all the other characteristics of the system will be influenced by this decision.

Taking into account all the advantages and disadvantages of the different architectures in Section 2.3.2.3 and the proposals presented in Chapter 3, we decided not to use the centralized or partially distributed approaches in our proposed naming system, thus avoiding the potentially significant overhead of name server election and synchronization mechanisms. We opted instead for the fully distributed architecture because we believe it is indeed the most natural fit for a MANET because of

its lack of infrastructure and adaptability to frequent changes in topology. Furthermore, it makes more sense that every node be equal and perform the same functions instead of burdening specific nodes with extra name serving tasks. The exception to this rule should be the nodes that are also connected to the Internet, which should have the extra functionality of acting as gateway nodes (see Section 2.3.2.2). We believe this to be a sensible exception since the connection to the Internet is a factor that clearly differentiates these nodes from the other ones.

Recall, however, that scalability and performance were the main drawbacks of a fully distributed architecture. Mainly due to its need of heavily relying on flooding mechanisms, which can overwhelm the network with message overhead. Our choice of the fully distributed approach hinges on the argument that these disadvantages can be mitigated if we can manage to perform name resolution without imposing such an excessive overhead. To achieve this, the naming protocol should take into consideration the underlying routing protocol and take full advantage of it in order to reduce the additional overhead of name resolution. The ways in which we can take advantage of the routing protocol, depend on the type of routing it implements. Therefore, the naming protocol should be different depending on whether we are dealing with reactive or pro-active routing.

In light of the above, we divided our proposal into two different protocols, explained in sections 4.2 and 4.3.

4.2 Pro-active Routing

Unlike Englestad's [17] integration of name resolution in reactive routing protocols, none of the proposals presented in Chapter 3 consider taking advantage of an underlying pro-active routing protocol in an ad hoc network. We decided to pursue this task and devise a protocol that could achieve this. We called it Name System for Pro-active Routing-Enabled Ad Hoc Networks (nameSPREAD).

The way in which pro-active routing is implemented and the types of messages used, can vary wildly from protocol to protocol, which means that any integration would have to be protocol-specific (much like Englestad's [17] proposal for reactive protocols). However, a protocol-independent solution (i.e., not integrated with the routing protocol) in the context of pro-active routing might not be as much of a problem as it would be in a reactive context (where it would almost certainly have to resort to flooding techniques). The main advantage of nameSPREAD is that it can pro-actively distribute names throughout the network without ever having to flood any messages (although the underlying routing protocol may use flooding, nameSPREAD itself does not).

In keeping with the fully distributed architecture spirit, nameSPREAD uses a flat namespace and allows nodes to choose their own names. As we saw in Section 2.3.2.2, allowing arbitrary names in order to keep them user-friendly, may lead to conflicts due to the chosen names being too simple. This problem is addressed in Section 4.2.6.

4.2.1 The Key Concept

As explained in 2.2, in pro-active routing protocols, every node keeps a route to each of the other nodes in the network. To achieve this, a node will exchange messages with its neighbors informing them of itself and of its other neighbors. In this way, the routing information is spread throughout the network. So when a node receives a route to a certain destination from one of its neighbors, it caches it in its routing table by associating the destination to the neighbor from which the route was received (the next hop), which can be observed in Figure 4.1.

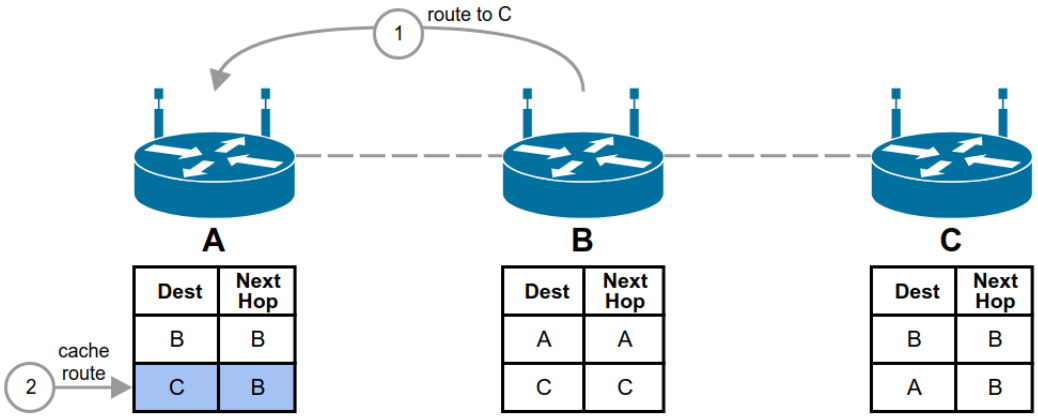

Figure 4.1: B announces to A that it knows a route to node C

The main idea behind nameSPREAD is that after adding a new route to the routing table, a node can ask the next hop of that route for the name of the destination in question. If every node does this, the names much like the routes – will be pro-actively spread as well (hence the name of the protocol).

Using Figure 4.2 as an example, C announces itself to B (1) which caches the route (2), asks C for its name (3) and then stores the response (4) in its local Domain Name System (DNS) cache (5). When A receives a route to C from B (6), it caches the route (7), asks B for C's name (8) and then stores the response (9) in its local DNS cache (10).

In this manner, every node will eventually know the names of all the other nodes, without

resorting to flooding. In fact, all that is needed to know a name of a node is to unicast a Name Resolution Request (NREQ) to the next hop towards that node and receive a unicast Name Resolution Reply (NREP).

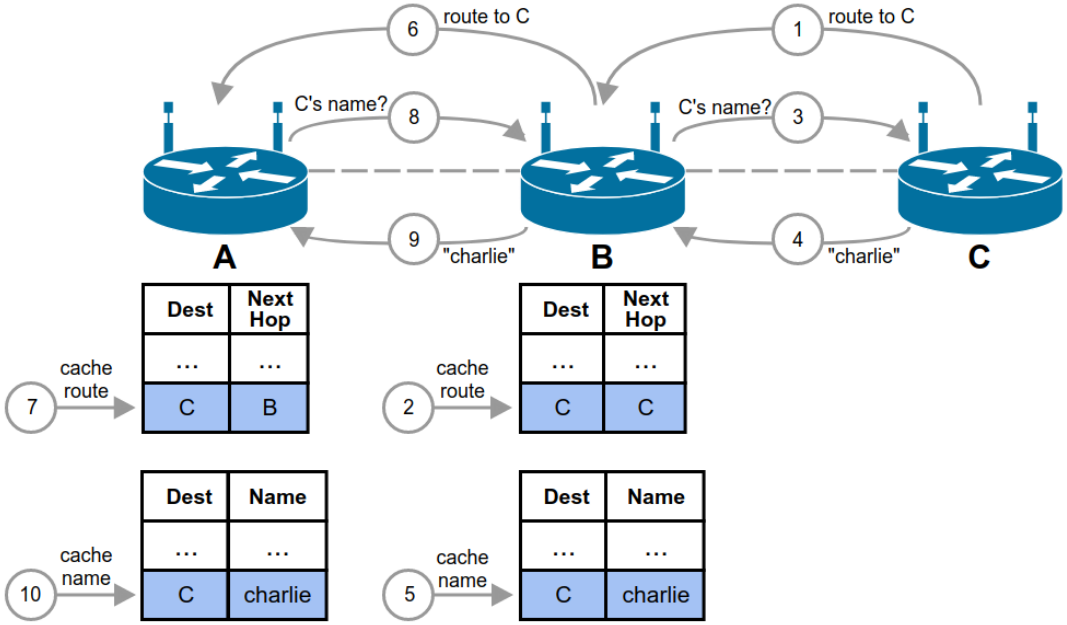

Figure 4.2: A node's name being spread along with its route

It is important to note that nameSPREAD caches the names it learns in the local DNS cache. Whenever a node needs to resolve a name, DNS will always check there first. Which means that nameSPREAD is completely independent and does not impose any modifications on other existing protocols.

4.2.2 Inner Modules and Their Tasks

This section provides a high-level view of the modules that comprise the nameSPREAD protocol. There are two main modules that grant it the functionalities that are needed to implement the process illustrated in Figure 4.2. First, it needs to be able to detect when a new route has been added to the routing table and second, it must be able to send and receive NREQs and NREPs. Bellow is a general description of the main nameSPREAD modules and the functionalities with which they are tasked.

The first module is the **Route Watcher**, whose job is two-fold: continually watch the routing table for changes and send NREQs to the neighbors from which new routes have been learned. Before sending the NREQ, the Route Watcher will also register it in a special table for pending requests (we will describe this in detail in the next section).

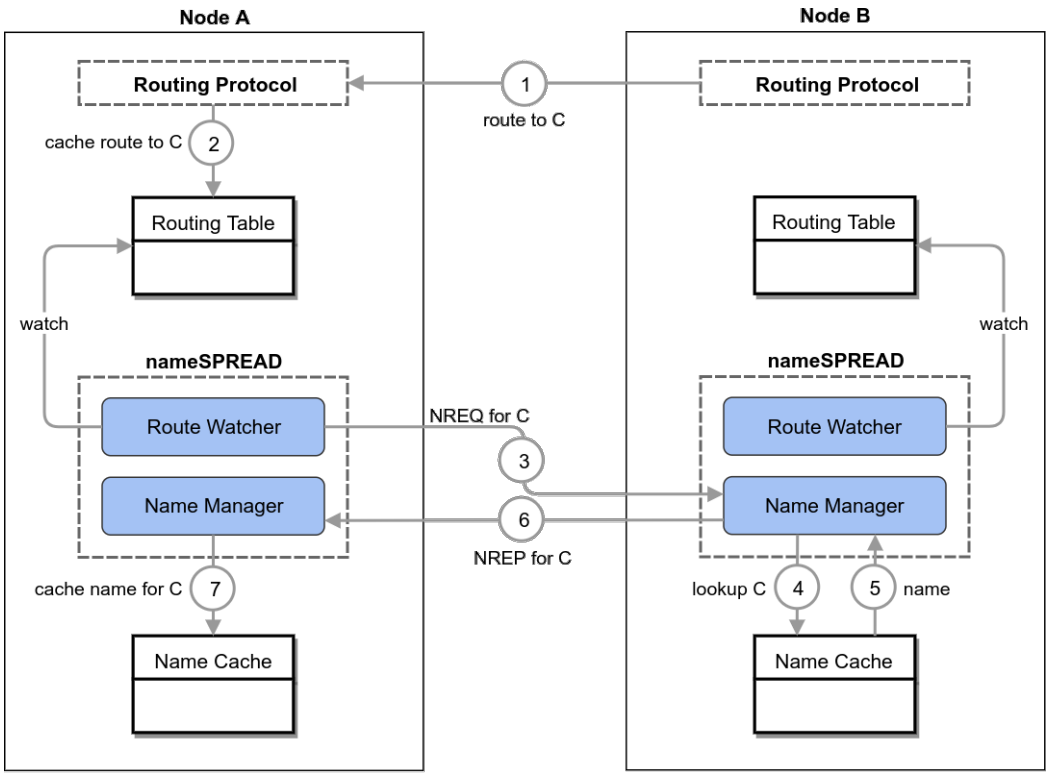

Figure 4.3: The nameSPREAD modules from different nodes interacting

The **Name Manager** is the other main module of nameSPREAD. Its essential tasks are to receive NREQs and NREPs and to send NREPs. That is, it must be able to receive name requests from its neighbors' Route Watchers and respond with the appropriate name, which will be receive by the neighbors' Name Managers. In order to respond to requests and to store new names, the Name Manager also has local DNS cache lookup and update functionalities.

In Figure 4.3, we can see that when node A's Route Watcher detects that a new route has been added to the routing table, it sends an NREQ to node B's Name Manager, which retrieves a name from the local DNS cache and sends it in the form of an NREP to node A's Name Manager, which finally caches it in its own DNS cache. As explained in the previous section, by the time A receives the route to C, B has already asked C for its name and so B can immediately respond when A asks it for C's name. But what happens if B receives an NREQ from A, but has not yet received the name from C? To deal with this situation, nameSPREAD uses a table to store NREQs to which it will respond as soon as it knows the answer. This is explained in the next section.

4.2.3 Name Cache Management

In a mobile environment, the local cache must constantly be updated to make sure that it reflects the known topology of the network and that it does not contain names for nodes that are either no longer in the network or that simply moved to another area of the same network. Both the Route Watcher and the Name Manager can play a part in keeping the local name cache fresh.

If we assume that any given node will always have the same name, then we consider the following name cache mechanism to be sufficient: when a route is lost (due to it not being refreshed by the routing protocol), the Route Watcher must remove from the local cache the name for the destination address of that route. However, because we are dealing with mobile nodes, the loss of a route might just be the consequence of an intermittent wireless connection, which means that the lost route might be available again fairly quickly and the premature removal of the name would have been unnecessary. Thus, as an improvement, instead of removing a name from the local cache as soon as the associated route is deleted, the Route Watcher could wait for a predetermined amount of time and then check if the route is still not available. Only then, would the name be removed from the cache.

If, on the other hand, we allow for the possibility of nodes changing their names while in the network, then the process described above would not work because it is dependent on detecting route losses. If a node changes its name without leaving the network, the other nodes will not lose the route to it and will therefore never ask for the new name. To avoid this problem, whenever the Name Manager caches a new name, it will associate a Time to Live (TTL) to the cache entry. When the TTL expires, if a route to the node is still active, a new NREQ will be sent to that node and the response will either cause a name update to be performed (if the node has changed its name) or no further action will be taken (in the case that the name remains the same).

4.2.4 The Pending Name Requests (PNR) Table

Thus far, we have established that Name Manager is the module that responds to incoming NREQs. But consider the scenario in Figure 4.4. After D announces to A, B and C that it knows a route to E, they are going to request E's name from D.

Now suppose D has not yet received the name from E. This means that B cannot respond to the NREQs because it does not know the answer yet. This is where the Pending Name Requests (PNR) table comes in. The PNR table is a key structure in nameSPREAD inasmuch as it allows a node to temporarily store name requests from other nodes (and from itself) to which it will respond once it

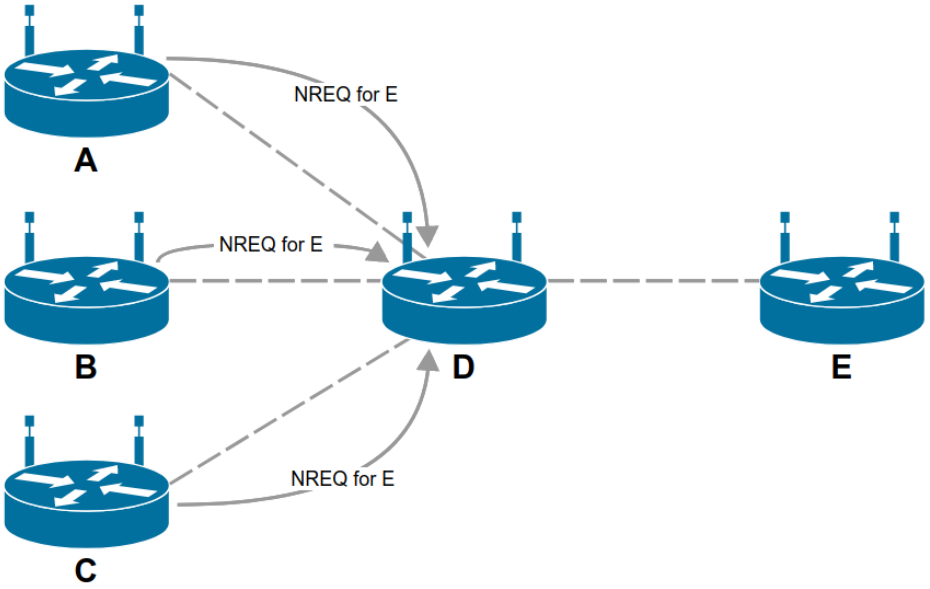

Figure 4.4: Multiple NREQs for the same name being received by one node

has learned the name of the node to which those requests refer.

To explain how the PNR table is used in nameSPREAD, let us continue with the example from Figure 4.4. As explained is Section 4.2.1, after D's Route Watcher has learned about the route to E, it will send an NREQ to E. However, before sending the request, it must register it in the PNR table by associating E's address to a list of nodes that have requested E's name. This can be seen in Figure 4.5 (a). Initially, this list only contains D because D's Name Manager has not yet received any NREQs for E and so, for now, D itself is the only known requester.

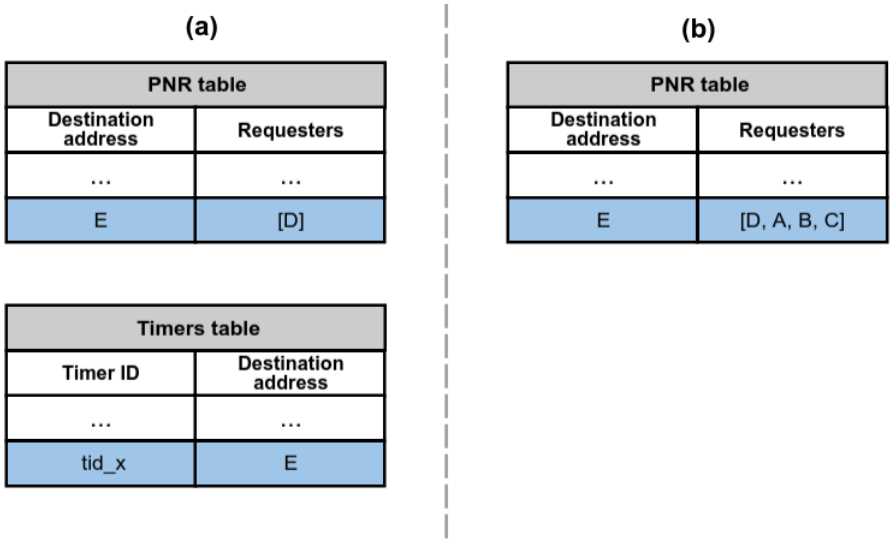

Figure 4.5: (a) new entries in D's PNR and Timers tables; (b) update of requesters list for E

When a new NREQ for a neighbor's name is registered in the PNR table, a new timer is started. The ID of the timer is stored in a table along with the address to which the NREQ refers (E's address). This table is another important structure of the nameSPREAD protocol because it allows the retransmission of NREQs that never got answered.

As we supposed earlier, if D's Name Manager is still waiting to know E's name when it receives the NREQs from A, B and C, then it can add them to the list of requesters for E (Figure 4.5 (b)). If an NREP is received before the timer expires, then D's Name Manager will send NREPs (containing E's name) to all the nodes in the requesters list for E's address. Of course, one of the requesters is D itself, in which case D will simply save the name in its local DNS cache. After this, D removes the related entries from the Timers and PNR tables.

If the timer expires before an NREP is received, D will retransmit the request to E. Note that A, B and C also created PNR entries (but not Timers entries) in their own tables for destination E. These entries will only be removed when an NREP is received or when the route to E is lost. This is due to the fact that, if a route to E is available through D, then it means that D also has a route to E, which means that a name for E can still be obtained.

4.2.5 Protocol Operation

Now that we have covered all the pieces that compose the nameSPREAD protocol, let us put them together to form a general diagram of its operation. Figure 4.6 is a flowchart that represents the workflow of both the Route Watcher and the Name Manager. It serves as a quick and easier reference for how the protocol works as it provides a visual representation of the modules' functions and interactions with both the PNR and the Timers tables.

All the processes depicted have been explained in previous sections. The only thing that warrants an explanation is the dotted line connecting the Route Watcher to The Name Manager. This line's purpose is to illustrate that an NREP received by the Name Manager is always received in the context of an NREQ that was sent previously by the Route Watcher to the node that advertised the route to the node whose name is in the NREQ. A node's Name Manager will never receive an NREP for a name for which it did not ask.

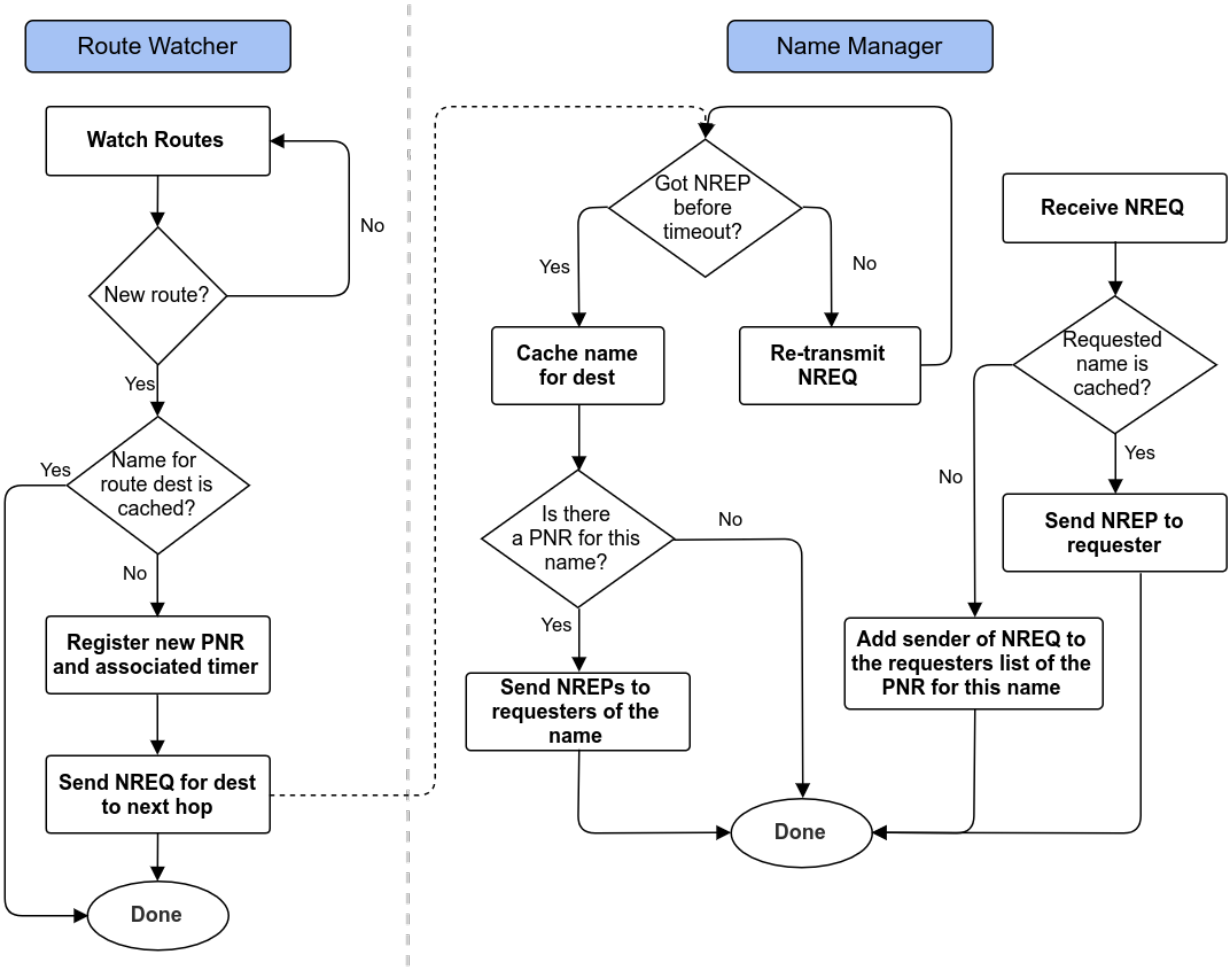

Figure 4.6: Wworkflow of nameSPREAD modules

4.2.6 Open Questions

Due to time constraints, some features of nameSPREAD were left out of the implementation of the protocol (described in 5). The following are suggestions of how these features could be implemented and integrated into nameSPREAD.

Name conflict detection/resolution: In nameSPREAD, nodes are free to choose their own names. This might lead to different nodes choosing the same name, which is not desirable as it can cause confusion as to whether we really are communicating with the node we want or just another one that happened to choose the same name. To solve this problem, whenever a node receives a name for an address which it already had previously cached for a different address, the node would warn the conflicting destination to change its name. However, the conflicting destination might already have sent the name to other neighbors that did not detect the conflict and therefore cached it. The conflicting node would then choose a new name and warn its neighbors with a Name Conflict (NC)

message. Whenever a node receives an NC message, it updates the cached name for the address indicated in the message. It would also forward the NC message to its neighbors. Eventually, all the nodes that had received the conflicting name will have been warned and updated their caches accordingly.

Notice that we are not considering the possibility of two different nodes having the same address. We believe that the naming protocol should not have to concern itself with that matter and that it should be dealt with by underlying protocols (i.e., by the routing protocol [58] [60]).

Network merging detection: The main problem with two networks merging, is that there might be nodes from each network that are using the same name. Although the name conflict resolution mechanism described above would work to solve this problem, it would not be fair. Suppose a two-node network merges with a 10-node network. If the former detected a conflict, it would not make sense to make all the nodes in the bigger network to update their caches. Instead, the nodes in the smaller network should be the ones to do it.

To detect when two different MANETs have merged, the nodes should always maintain two values: a Partition ID (PID) and a network size value. The PID would be a random and sufficiently large number (to guarantee uniqueness) and the network size would be initiated to one upon initialization of the protocol in each node and updated when two different networks merge.

When two nodes come into contact, they tell each other the number of nodes in their respective networks. The node in the smaller network changes its PID to match the larger network's PID, changes its network size number to the sum of both sizes and notifies the other nodes in its network to do the same. If a name conflict is detected during the exchange of routes/names, the conflicting node in the smaller network is warned to change its name and follow conflict resolution process described previously.

Resolving outside names with DNS through gateway nodes: Upon initialization, if the protocol detects that the node has a working Internet connection, it will append *.gw* to its name. The protocol should also intercept DNS queries and, if the query is for an outside name, send the query to any node in its DNS cache whose name contains the *.gw* suffix. A gateway node will receive the query, send it to its DNS server and return the response to the node that originated the query.

4.3 Reactive Routing

In Chapter 3 we described Engelstad's [17] proposal for name resolution integrated in a reactive routing protocol. As we mention in 4.1, the naming protocol should try to take advantage of the routing protocol. In a reactive context, we believe that Engelstad's [17] concept is the one that makes the most sense. The proposal is generic enough to be applied in any reactive routing protocol because they all use Route Request (RREQ) and Route Reply (RREP) messages in one way or another, although the implementation should be protocol-specific because different protocols have different message formats.

The whole point of Engelstad [17] is to resolve names while not adding any extra flooding message and so name resolution is performed simultaneously with route discovery, by piggybacking NREQs on RREQs and NREPs on RREPs. An implementation of this name resolution concept should be specific to the routing protocol because if it was independent from the routing protocol, it would certainly have to rely on flooding to resolve names, which is what Engelstad [17] was trying to avoid in the first place.

Based on the concept from Engelstad [17], we instantiated a design of it using AODV, which is one of the most popular reactive routing protocols specifically designed for ad hoc networks [17]. We called it Naming Extension for AODV (NExtA). Because the basic concept was already explained in Section 3.1.2.2, we will limit the description of NExtA to some AODV details and the specific modifications that would have to be made to its original messages in order for it to support name resolution.

4.3.1 Extending AODV for Name Resolution

To understand some of the modifications to the AODV messages, there are some notions we should keep in mind. For instance, every AODV node maintains a routing table with the usual information (i.e., destination address, next hop address, etc), but also registers the last known sequence number for the destination of a route[1] and a list of precursors[2]. AODV nodes use four types of messages to communicate known routes to each other. The RREQ and RREP messages are used for the route discovery process and the Route Error (RERR) and HELLO messages, which are used for route maintenance.

[1] Sequence numbers are used by AODV to avoid routing loops when calculating a new route.
[2] Contains neighbor nodes to which an NREP was forwarded.

The following is a list containing one item for each AODV message type that needs to be modified, with a brief description of its purpose and suggested modifications (in bold font) to its original format.

- **HELLO:** messages are used to monitor connections to neighbors. They are periodically broadcast by every node to its neighbors. If a node fails to receive HELLO messages from a neighbor for some time, a broken link is detected and an RERR is sent to the list of precursors of the routes that use this link to get to the next hop.

  <dst_addr, **dst_name**, dst_seq#, hop_count, lifetime >[3]

  If a node inserts its own name in a HELLO message, the node that receives it can save the neighbor's address and name in its DNS cache. However, because HELLO messages are exchanged often, the overhead of including a name could be significant. As such, this could be an optional functionality, seeing that it is not essential.

- **RREQ** messages are broadcast by a node (the originator node) when it wants to discover a route to another node to which it does not yet know a route.

  <id, dst_addr, **dst_name**, dst_seq#, originator_addr, **originator_name**, originator_seq#, hop_count >

  The main idea is to perform route discovery *by name* instead of *by address*, allowing a node to communicate with another node even if does not know its address (i.e., it only needs to know its name). And so, the destination address could be left blank as long as the destination name is introduced in the RREQ. This way, name resolution and route discovery are performed simultaneously.

  Furthermore, providing the originator's name in RREQ messages, allows intermediate nodes (nodes that simply forward the request towards its destination) and the destination node to cache the originator's name. This would have to be tested in order to assess if the gain of caching in intermediate nodes is worth the overhead of including the extra name. It could be provided as optional functionality since it is not crucial that the originator's name be known either by the intermediate nodes or the destination node.

- **RREP** messages are sent in response to RREQs by a node that knows a route to the destination node (or the destination node itself).

[3] In HELLO messages, *dst* is the node that generates the message (i.e., not the one the that receives it).

<dst_addr, **dst_name**, dst_seq#, originator_addr, hop_count, lifetime >

If the node whose route is being requested includes its own name in the RREP, then the originator node will be able to cache that name in its DNS cache. Furthermore, intermediate nodes could also cache the name.

Notice that the only indispensable modification is the inclusion of the destination name in the RREQ. Imagine, for instance, that we choose not to use the option of including the originator's name in the RREQ. When a node (the originator) sends an RREQ, it would cause the destination node and all the intermediate ones to discover a route to the originator, but not the originator's name. Should one of the intermediate nodes want to communicate with the originator, it would not send an RREQ because a route to the originator is already known, which means that it would have no way of ever knowing the originator's name. To solve this problem, new messages — for specifically asking a node for its name and for responding to such a message — would have to exist. They would be similar to regular NREQs and NREPs, but sent in unicast, since the route is already known. These new messages are not included in Engelstad's [17] proposal, but since they are sent in unicast, they do not violate the original principal of trying to avoid the overhead of flooding.

With the suggested modifications to the original AODV messages, NExtA would be able to provide nodes with the names of those with whom they wish to communicate. Figure 4.7 provides a general view of the AODV protocol workflow with the addition of the functionalities that constitute the extension for name resolution.

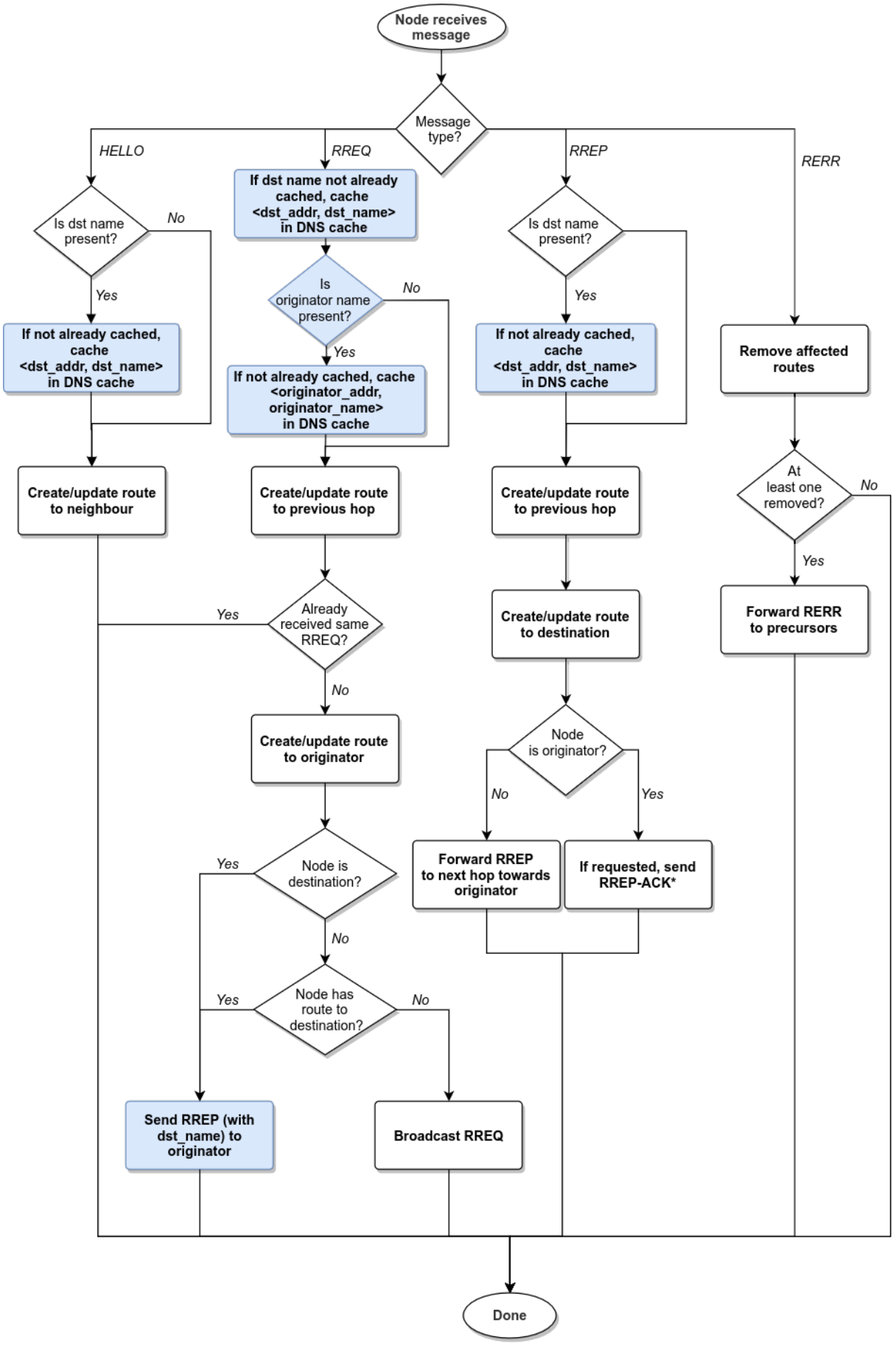

Figure 4.7: NExtA workflow upon message reception

Chapter 5

Implementation of nameSPREAD

Having conceived the two naming systems presented in Chapter 4, we proceeded to implement Name System for Pro-active Routing-Enabled Ad Hoc Networks (nameSPREAD) — our proposal for ad hoc networks that depend on a pro-active routing protocol. This implementation allowed us to put into practice the protocol design described in Section 4.2 so we could evaluate its performance (see Chapter 6).
In this chapter, we describe the implementation of the main Route Watcher and Name Manager modules that compose nameSPREAD, the hash tables used to implement the Pending Name Requests (PNR) and Timers tables, and the logging system that is used to record every relevant event.

5.1 Overview

We chose to implement nameSPREAD using C as the programming language given its efficiency (due to the thin layer of abstraction that separates it from actual assembly language).

As determined in 4.2.2, the Route Watcher needs to continually monitor the routing table and the Name Manager must always listen for incoming messages. At the start of our program, two separate threads are created to allow the two modules to run concurrently (Code Block 5.1).

The *main()* function starts by configuring the PNR and Timers hash tables (further described in Section 5.2). Then it initiates the two threads which trigger the execution of the major function of their modules: *watch_routes()* for the Route Watcher and *listen_for_msgs()* for the Name Manager. A third thread is also initiated which will launch the protocol's logging system. This is omitted from the code bellow, in the interest of clarity.

```c
int main(int argc, char *argv[]) {
    own_addr = argv[1];

    pnrs = cfuhash_new();
    timers = cfuhash_new();
    cfuhash_set_flag(pnrs, CFUHASH_FROZEN_UNTIL_GROWS);
    cfuhash_set_flag(timers, CFUHASH_FROZEN_UNTIL_GROWS);

    pthread_create(&tid1, NULL, listen_for_msgs, NULL);
    pthread_create(&tid2, NULL, watch_routes, NULL);
}
```

Code Block 5.1: The *main* fucntion of nameSPREAD

The source code of whole project is available at https://github.com/gflcampos/nameSPREAD.

5.2 PNR and Timers Hash Tables

As we explained in Section 4.2.4, to store information related to name requests that were sent to other nodes, we need a key-value data structure with which we can make the following associations:

NREQ destination address <-> List of requesters

Timer ID <-> NREQ destination address

We opted for the use of hash tables due to their efficient insertion and lookup processes. This way, we can register, lookup, update or remove pending name requests with ease and efficiency. We also need a first in, first out (FIFO) data structure — such as a linked list — to serve as the list of requesters for a given name. These data structures are available in the C library *libcfu* [47], which contains thread-safe implementations of both hash tables and linked lists.

As show in Code Block 5.1, after being created, the PNR and Timers hash tables are configured with the *CFUHASH_FROZEN_UNTIL_GROWS* flag, which means that they will only be rehashed when the number of entries divided by the number of buckets reaches the threshold of 0.75.

5.3 Route Watcher Module

The main loop of the Route Watcher module monitors the routing table for changes by using the *popen()* function to create a new process in which the command *ip monitor route* is executed. Whenever a change is made to the routing table, this command will print information to *stdout*, which is then parsed by the Router Module in order to assert when a route is added or removed. Another way of doing this would be to use a *netlink* socket [34] with *NETLINK_ROUTE* as the *netlink* family parameter, so we could receive routing and link updates from the kernel. This is actually what *ip monitor route* does, but we could use a *netlink* socket directly and avoid the *ip* command altogether. Due to time restrictions, we opted for the first approach, which is simpler and straightforward.

Whenever a route removal is detected, the Route Watcher checks if there is a pending name request for the destination of the lost route and, if it finds one, it is removed from the PNR hash table, as the route to that destination no longer exists. Furthermore, if a name for the destination of the lost route is cached, then the pending name request is removed from the hash table as well.

When a new route is detected, if a name for the destination of the route is not yet known (this is checked by reading the */etc/hosts* file) and a pending request for that name does not yet exist, the node registers a new request in its PNR hash table and sends a Name Resolution Request (NREQ) to the next hop of the new route. A *register_nreq()* function creates a new entry in the node's PNR table that maps the destination address of the NREQ to a new linked list that contains the address of the node itself. This way, should the node receive requests for this name, it knows it is already waiting to receive a response for that name and can simply add the new requester to the list.

The same function also starts a new timer and stores its ID in the Timers hash table along with the destination address of the NREQ. A *make_timer()* function defines the expiration time of the timer, the signal that should be generated upon expiration and the function that is executed to handle that signal. When the timer expires, the *SIGRTMIN* signal is generated, which triggers the execution of a *timeout_handler()* function. This function is passed the ID of the expired timer, which it can use to retrieve the destination address of the associated NREQ from the Timers hash table. Then, it checks if a pending name request for that address exists in the PNR hash table. If it does not, then it means that either a name for that address was already received or the route to that address no longer exists, in which case the timer is removed from the Timers table. Otherwise, the timer is reset and a new NREQ is sent.

5.4 Name Manager Module

The Name Manager module runs along-side the Route Watcher on its own thread. It continuously waits for messages on port 7891 of a User Datagram Protocol (UDP) socket. We chose to use UDP because of its smaller overhead of packet transmission in relation to Transmission Control Protocol (TCP). However, due to UDP's lack of reliability, some packets might be lost, which could result in some nodes never receiving the names of all the other available nodes. We solved this issue by retransmitting NREQs upon timer expiration, as we explained in the previous section.

When an NREQ is received, if the requested name is cached (in the */etc/hosts* file), the Name Manager immediately responds with an Name Resolution Reply (NREP) containing the requested name. Otherwise, it adds the requester to the linked list of requesters for that name in the PNR entry that itself had created earlier. Due to varying execution times in the different nodes, it might happen that a node receives a name request for a destination whose route it has already received but has not yet had time to register an NREQ in the PNR hash table. The *register_nreq()* function is prepared to deal with this possibility and so it creates a new pending request if one does not already exist. An NREQ is sent only when the node adds itself to the list of requesters for the name in question.

When the Name Manager receives an NREP, it caches the name (by writing it in the */etc/hosts* file) and then forwards the same NREP to every address in the linked list of requesters of the corresponding pending name request (which is then removed from the PNR hash table).

5.5 Logging System

nameSPREAD also includes a logging mechanism that records every relevant event such as the addition or removal of routes, receiving or sending of NREQs or NREPs, caching or removal of names, NREQ timeouts, etc. Every event is registered along with a timestamp, as shown in Code Block 5.2.

```
...
[25-08-2017 15:59:01] [ROUTE+] New route added: 10.0.0.48 via 10.0.0.92
[25-08-2017 15:59:01] [TIMER+] Started timer 6 for NREQ of 10.0.0.48
[25-08-2017 15:59:01] [PNR+] New PNR for 10.0.0.48 with localhost as requester
[25-08-2017 15:59:01] [NREQ->] Requesting name for 10.0.0.48 from 10.0.0.92...
...
[25-08-2017 15:59:05] [<-NREQ] 10.0.0.33 wants name for 10.0.0.48
[25-08-2017 15:59:05] [PNR++] Added 10.0.0.33 as requester for 10.0.0.48 in PNR
...
[25-08-2017 15:59:07] [<-NREP] received name 'sta48' for host 10.0.0.48
[25-08-2017 15:59:07] [NAME+] Cached name for host 10.0.0.48
[25-08-2017 15:59:07] [NREP->] Sent name 'sta48' to waiting requester 10.0.0.33
...
[25-08-2017 15:59:24] [TIMEOUT] Timeout on NREQ for host 10.0.0.75 (timer ID: 2)
[25-08-2017 15:59:24] [TIMER++] Reset timer 2 for NREQ of 10.0.0.75
[25-08-2017 15:59:24] [NREQ->] Requesting name for 10.0.0.75 from 10.0.0.6...
...
```

Code Block 5.2: Excerpt from a nameSPREAD file log

Chapter 6

Experimental Evaluation

This chapter focuses on the testing of our name resolution protocol for pro-active ad hoc networks: Name System for Pro-active Routing-Enabled Ad Hoc Networks (nameSPREAD). In Section 6.1, we present our options concerning an implementation of Optimized Link State Routing (OLSR) — the pro-active routing protocol used for our experiments, followed by Section 6.2 where we describe Mininet-WiFi [15], which is the network emulation software we used to create the necessary conditions for the evaluation of the protocol in different scenarios. The main objective of the tests is to assess how nameSPREAD performs in different situations and how much of a message overhead it imposes. We conclude the chapter by examining the results gathered during the testing sessions.

6.1 OLSR.org Network Framework (OONF)

As far as the routing protocol is concerned, we opted for OLSR since it is one of the most popular pro-active routing protocols used in mobile ad hoc networks in recent years [8].

Table 6.1: OLSR implementations

	OLSR version	Developed by	Last update	Linux	ns-2	OMNeT++
OONF	1 & 2	OLSR.org	12/09/2017	✓	–	–
NRL-OLSR	1	US Naval Research Laboratory	10/08/2007	✓	✓	–
UM-OLSR	1	Francisco J. Ros (University of Murcia)	05/08/2015	–	✓	✓

We gathered the three most relevant implementations of OLSR (see Table 6.1) that we were able to find. OONF is by far the most actively developed implementation and the only one that includes version 2 of OLSR. It is only intended for use in Linux-based operating systems and not network simulators but, since we decided to use Mininet-WiFi — which runs real code over the Linux kernel — OONF was the clear choice.

6.1.1 NetJSON Info

OONF includes many plugins that implement different network functionalities. One of these plugins was of particular interest for our needs: the NetJSON Info plugin [46]. NetJSON [11] is a data interchange format for describing network properties such as nodes' routing tables or the network topology, using the JavaScript Object Notation (JSON) format.

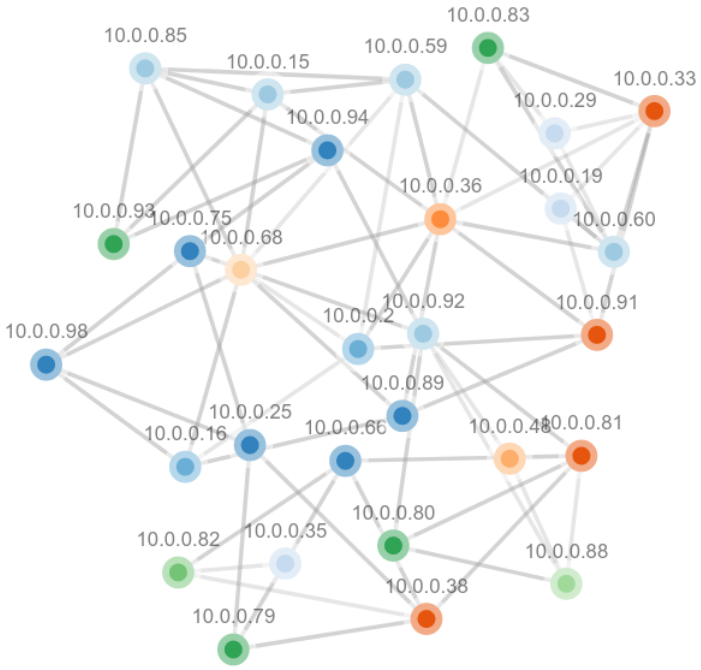

Figure 6.1: Graph of larger cluster in testing scenario 4 (see Section 6.3) generated by *netjsongraph.js*

We used NetJSON Info to determine the number of existing links in each test scenario (see Section 6.3) so we could calculate the network density. By querying one node in a cluster via *telnet* on port 2009 with the command *netjsoninfo graph*, we obtain a JSON object containing all the links in that cluster (which can also be used to generate a visual representation of that cluster with *netjsongraph.js* [21], as illustrated in Figure 6.1). However, NetJSON Info only considers some of the links to be bidirectional, but because we use a deterministic propagation model (Section 6.2.1),

all the links are bidirectional. To overcome this issue, we parsed the JSON object with a Python script (see Appendix B) that, given one of the test scenarios, counts the total number of links in each cluster of nodes and calculates the network density.

6.2 Mininet-WiFi

To evaluate the performance of the protocol we need to really push it to the limit by experimenting with a large number of network nodes. Since it would be logistically infeasible to do large scale experiments with real hardware nodes, we decided to use simulation/emulation software to create the necessary conditions for testing nameSPREAD. As we mention in Section 1.2, we considered using ns-2/3 [22] or OMNeT++ [59] to simulate our testing scenarios but we eventually settled on the network emulator Mininet-WiFi [15].

While simulators and emulators both try to mimic some other system, an emulator will also implement the inner workings of that system rather than just its behavior. This allowed us to develop nameSPREAD in conditions as close as possible to what would be the real conditions where the protocol would run — meaning that the same nameSPREAD code used in Mininet-WiFi, can also run in any Linux-based operating system without any modifications.

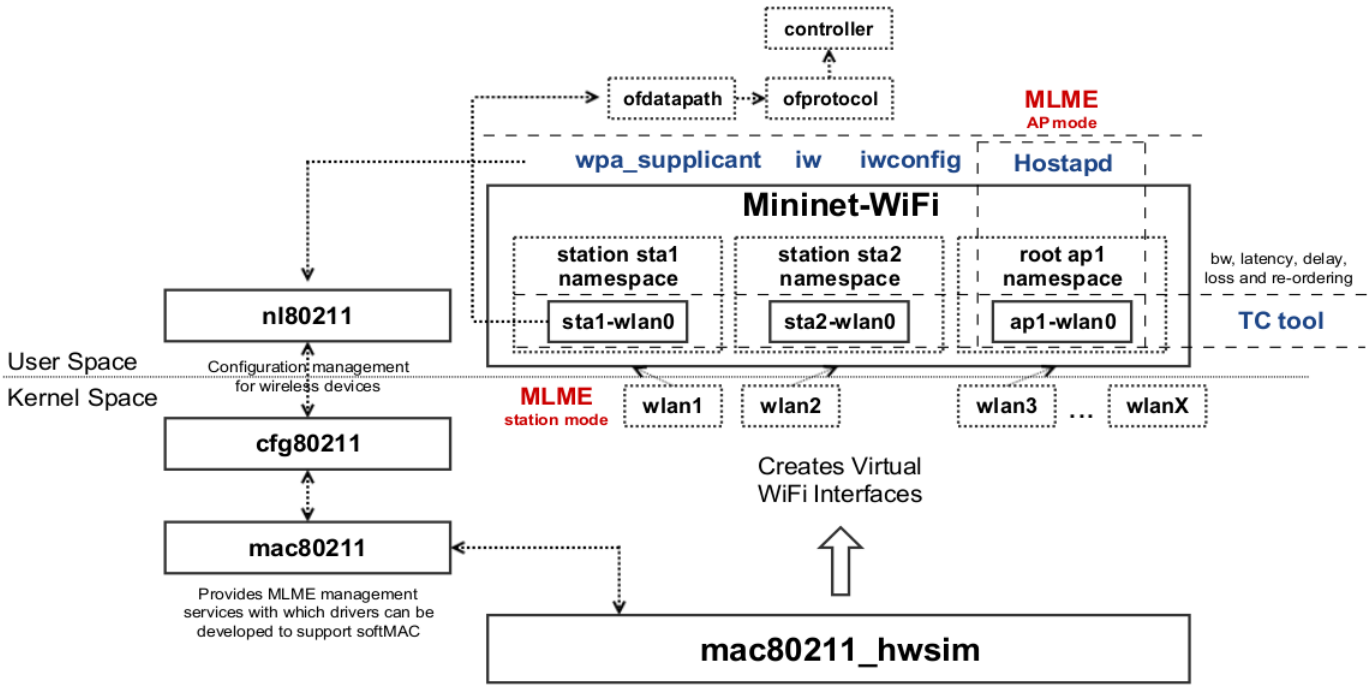

Figure 6.2: Mininet-WiFi components [16]

Mininet-WiFi is a fork of Mininet [37], which is a network emulation system (implemented in Python and C) that creates virtual networks running real Linux kernel code. By making use of Linux network namespaces, Mininet allows the hosts in the emulated network to have their own set of interfaces (with their own IP addresses) and routing tables (Figure 6.2). Each host runs as a bash process on the same machine but in different network namespaces.

Mininet also provides a Python API which can be used to create scripts to configure and start a virtual network. Moreover, a Python client with a command line interface is available for real-time manipulation of the network.

Mininet-WiFi extends Mininet by adding the possibility of using WiFi stations and access points as hosts on an emulated network. It makes use of the *mac80211_hwsim* Linux kernel driver [39] to create virtual IEEE 802.11 wireless interfaces. However, *mac80211_hwsim* simulates perfect frame transmission conditions by default (e.g., no packet loss or corruption). In order to simulate the wireless medium in a more realistic way, Mininet-WiFi uses either the Linux Traffic Control (TC) [35] utility, which is included in the *iproute2* package [33], or *wmediumd* [14] — a tool developed as an alternative for the default forwarding mode of *mac80211_hwsim*. To further simulate wireless conditions, Mininet-WiFi provides different wireless signal propagation models [36, Chapter 2] (e.g., Friis, Two-Ray Ground, Log-Distance, etc.) to define how radio waves should propagate as a function of distance, frequency, antenna gain and other properties.

Lastly, nodes in a Mininet-WiFi network can be statically positioned or move according to different mobility models [10] (e.g., Random Waypoint, Random Direction, Gauss-Markov, etc.). The movement of the nodes can be seen in real-time in a graph plotted by Mininet-WiFi.

6.2.1 Setup

To create the environment for our test scenarios, we wrote a Python script for Mininet-WiFi to configure our network. Link and network layer addresses are automatically assigned to the wireless stations by Mininet-WiFi from the default address pools:

IP pool: 10.0.0.0/8

MAC pool: 02:00:00:00:00:00 — 02:00:00:00:FF:FF

As shown in Code Block 6.1, the wireless interfaces of each station started in ad hoc mode and use frequency 2.437 GHz (channel 6) with a transmission power of 20 dBm.

```
mininet-wifi> sta2 iwconfig sta2-wlan0
sta2-wlan0  IEEE 802.11abgn  ESSID:"adhocNet"
            Mode:Ad-Hoc  Frequency:2.437 GHz  Cell: 02:CA:FF:EE:BA:01
            Tx-Power=20 dBm
            Retry short limit:7   RTS thr:off   Fragment thr:off
            Encryption key:off
            Power Management:off
```

Code Block 6.1: iwconfig output on a Mininet-WiFi ad hoc station

As we explained in the last section, all stations have their own network namespace. However, they share the host's filesystem and process ID space. Recall from Chapter 5, that nameSPREAD needs to have read/write access to the *hosts* file of the machine on which it is running, which means that each station must have its own *hosts* file.

To isolate the stations' *hosts* files from each other, we use the Linux *mount* command with the *--bind* option that allows us, for each station, to bind a different file to the */etc/hosts* pathname (see Code Block 6.2). This way, when station 1 reads from */etc/hosts*, it is actually reading from */tmp/mininet-wifi/hosts/hosts-sta1*.

```
for i in range(1, nStations+1):
    hostsCmd = "mount --bind /tmp/mininet-wifi/hosts/hosts-sta%d /etc/hosts" % i
    net.stations[i-1].cmd(hostsCmd)
```

Code Block 6.2: Isolation of */etc/hosts* files for each station

Although we ended up not using mobility for the test scenarios (for reasons explained in Section 6.3), we initially envisioned the nodes as mobile devices carried by people at an average walking speed of 1.5 m/s [9]. To simulate this, Random Waypoint is the most appropriate mobility model in that it makes each node choose a random point on the available area and move towards it at a certain speed (selected randomly from a defined interval). After arriving at that point, the node will wait for a random amount of time (Mininet-WiFi defines 100 seconds as the maximum waiting time) and then starts the process again. We believe this to be the closest model to the walking pattern of an actual person (although it does not consider natural obstacles that might exist). Furthermore, we used the default antenna height of 1 meter (from the ground) and gain of 5 dB.

```
print("*** Adding mobility")
net.startMobility(model='RandomWayPoint', max_x=200, max_y=200, min_v=1.5,
    max_v=2)
```

Code Block 6.3: Selection of Random Waypoint as the mobility model for all stations

As for the wireless medium simulation, we used *wmediumd* which is the appropriate choice for ad hoc networks as per the Mininet-WiFi manual [16] and the Log-distance propagation model. At first, we wanted to use the probabilistic Log-normal Shadowing model but found out (through debugging and email discussion with the Mininet-WiFi maintainer) that it was not correctly implemented and so we eventually settled for the deterministic Log-distance model.

```
print "*** Configuring propagation model"
net.propagationModel('logDistancePropagationLossModel', exp=4)
```

Code Block 6.4: Selection of Log-distance as the propagation model for all stations

The Log-distance propagation model calculates the path loss that a radio signal will experience over a distance. The formula of this model is as follows:

$$PL_{r,dB}(d) = PL_{dB}(d_0) + 10\ \alpha\ \log\left(\frac{d}{d_0}\right) \quad [36, p.12]$$

Where

$PL_{r,dB}(d)$ is the received power (dB)

$PL_{dB}(d_0)$ is the path loss (dB) between the sender and the receiver over a distance d (m)

d_0 is a close-in reference distance (1 m in Mininet-WiFi's implementation of this model)

α is the path loss exponent

Environment		α
Outdoor	Free space	2
	Shadowed urban area	2.7 to 5
Indoor	Line-of-sight	1.6 to 1.8
	Obstructed	4 to 6

Table 6.2: Typical values for the path loss exponent [54]

Area	Trees/m^2	α
High density	0.028	4.2
Medium density	0.021	3.9
Low density	0.009	1.8

Table 6.3: Path loss exponents values in different tree density areas [51]

The path loss exponent can be used to obtain different path loss values to simulate different environments. Looking at tables 6.2 and 6.3, we can conclude that by choosing 4 as the path loss exponent to our log-distance propagation model, we can simulate both an urban area and a medium/high tree density forest area.

The range of our wireless stations is calculated by Mininet-WiFi upon network initialization and is the same for all stations (since Log-distance is a deterministic model) and constant throughout the lifetime of the network. Based on our path loss exponent and other default properties of the virtual stations, the wireless range determined by Mininet-WiFi is 33 meters.

Finally, we launch OONF and nameSPREAD on each station and enable the *hwsim0* interface (an interface created by Mininet-WiFi on the host machine that allows us to monitor all the traffic in the network).

```
print("*** Starting OLSR and NameSPREAD on each station")
for i in range(1, nStations+1):
    addr = net.stations[i-1].params['ip'][0]
    net.stations[i-1].cmd("<namespread_path>/src/namespread %s &" % addr)
    net.stations[i-1].cmd("setsid <oonf_path>/build/olsrd2_static sta%d-wlan0
        lo &>/dev/null &" % i)

os.system("ifconfig hwsim0 up")
```

Code Block 6.5: Running OLSR and nameSPREAD and enabling *hwsim0*

The full Python script is available in Appendix A.

6.3 Testing

With the emulation software ready, we set out to test nameSPREAD by designing different scenarios and observing how the protocol reacts to them. As mentioned in the previous section, all scenarios are trying to simulate a substantially obstructed outdoor setting. A predefined number of stations are positioned randomly on differently sized areas so we can produce different network densities. We decided not to introduce mobility for the purposes of testing the protocol because it allows us to have more control over the density. By fixing the position of the stations, the density will remain constant, which will produce more measurable results.

Table 6.4: Testing scenarios

Scenario	Area (m^2)	Dimensions (m)	Number of stations	Stations/m^2
1	10000	100x100	–	–
2			10	0.001
3	100000	316x316	30	0.0003
4			100	0.001
5	1000000	1000x1000	100	0.0001
6			–	–

We defined the six scenarios displayed in Table 6.4. The basic idea was to test the protocol for different network densities in different areas. For each area we have a sparse and a dense case, with the exception of the sparse case for an area of 10000 m^2 (scenario 1) and the dense case for an area of 1000000 m^2 (scenario 6), which were not actually tested. We considered the first scenario to be irrelevant because it would contain less than 10 stations, which would not be a challenge to the protocol. As for scenario 6, it would have contained around 300 stations, but the *mac80211_hwsim* driver imposes a limit of one hundred 802.11 radios.

We also determined the network density for each tested scenario. Network density is defined as the proportion between the number of connections that exist between nodes in a network (actual connections) and the maximum number of connections that could possibly exist (potential connections) [53]. Let *n* be the number of nodes in a network, then:

$$Network\ Density = \frac{Actual\ Connections}{Potential\ Connections} \qquad Potential\ Connections = n(n-1)$$

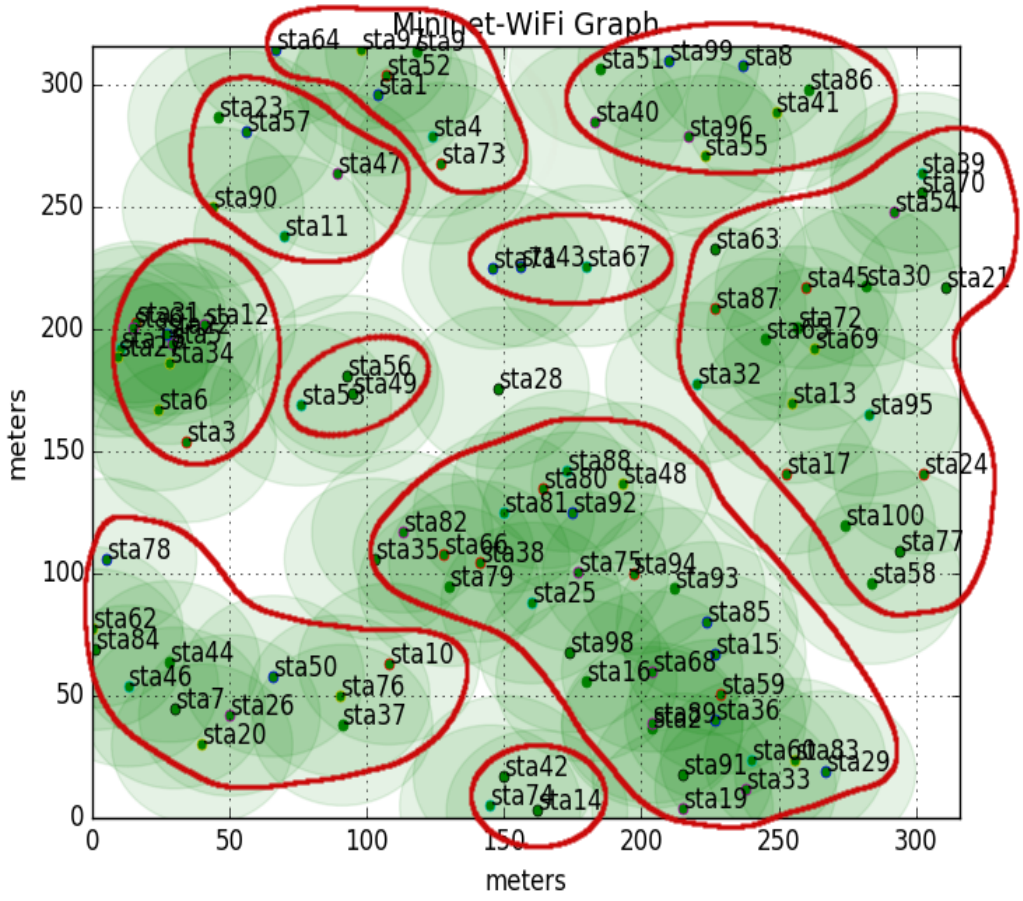

Figure 6.3: Clusters (circled in red) of test scenario 4 (see Appendix C for the other scenarios)

To calculate the density of a scenario, we must first determine how many links exist in every cluster of said scenario, so we can compute the total number of links in the graph. After identifying the clusters in the scenarios (e.g., Figure 6.3), we use the information obtained with OONF's NetJSON Info plugin (see Section 6.1.1), to calculate the densities using the Python script in Appendix B, which implements the above formulae. The results can be seen in Table 6.5.

Table 6.5: Network densities for the different test scenarios

Scenario	Number of nodes	Potential connections	Actual connections	Density
2	10	90	30	0.3333
3	30	870	48	0.0552
4	100	9900	340	0.0343
5			28	0.0028

6.4 Results

In every scenario, we measured the time it took for all the nodes to learn all the names they could possibly know (i.e., the names of the other nodes in the same cluster) — henceforth referred to as the *Elapsed Time*. Using Wireshark, we registered the number of messages generated by both nameSPREAD and OLSR during this time, so we could assess the real impact of introducing support for name resolution. We ran nameSPREAD ten times on each scenario. Table 6.6 shows the average of the ten sets of values gathered from the tests.

Table 6.6: Number of messages generated by OLSR and nameSPREAD

		OLSR		nameSPREAD	
Scenario	ElapsedTime (sec)	# messages	Size (kB)	# messages	Size (kB)
2	10.3	448.5	216.9	180	18.1
3	17.9	1809	787	204	20.7
4	246.4	95773	47812.5	3850.5	391.2
5	14.8	2682.5	802.3	64	6.5

We can see that scenario 4 is by far the one that took the most time, due not to the number of nodes (100), but to the high number of actual connections (as defined in the previous section). In scenario 5, there are also 100 nodes, but as there are only 14 actual connections, the Elapsed Time is much lower.

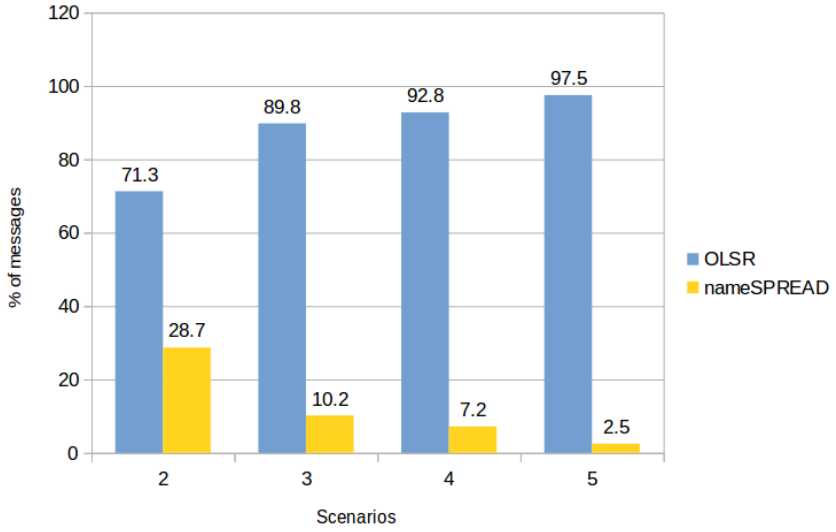

Figure 6.4: Percentage of nameSPREAD and OLSR messages

Figure 6.4 show the percentage of messages of each protocol for the five scenarios. We see in Table 6.6 that as the number of nodes grows, so does the amount of OLSR and nameSPREAD messages, but number of messages grows at a much higher rate in the case of OLSR, which leads to a growing percentage of messages from scenario to scenario.

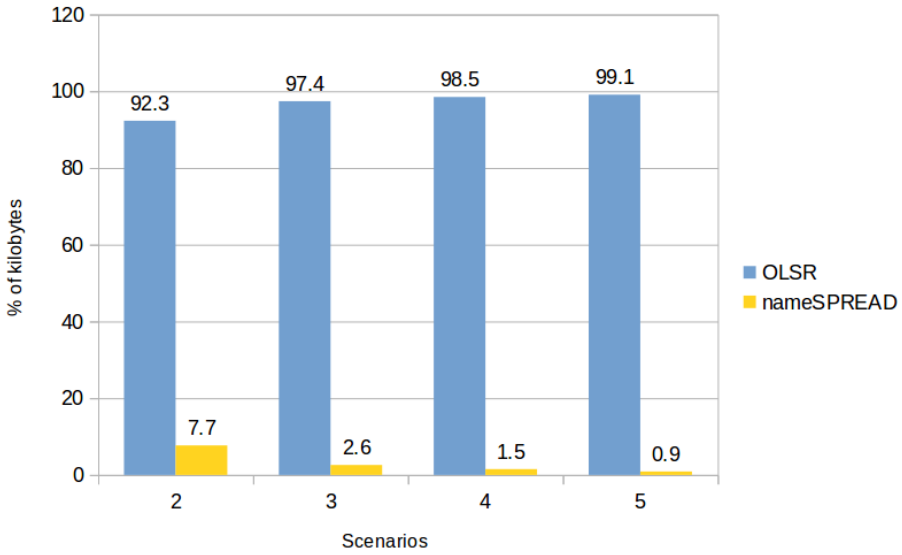

Figure 6.5: Percentage of kilobytes of nameSPREAD and OLSR

An obvious consequence of a smaller amount of generated messages from nameSPREAD, is that the total size of bytes sent is also smaller. Figure 6.5 show the same effect as before but with an even more accentuated difference. In all scenarios, less than 8% of all captured bytes come from nameSPREAD. Even in scenario 4 — which has the highest number of connections — nameSPREAD only needed 391.2 kilobytes to provide all the nodes with the necessary names.

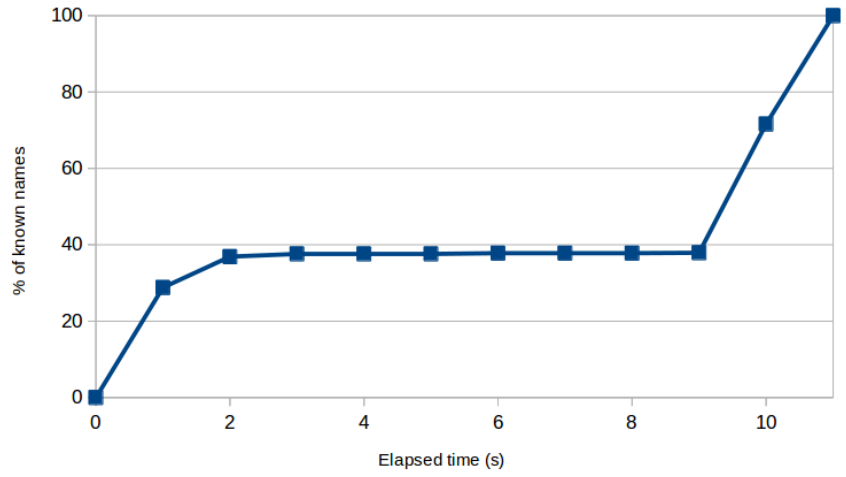

Figure 6.6: Cumulative distribution function for scenario 2

We also calculated the cumulative distribution functions for each scenario, by parsing the log files with the bash script in Appendix D. At each give second of the Elapsed Time, we can observe the total number of learned names in the entire network. For instance, in scenario 3 (Figure 6.7), 95% of names had been learned after just 15 seconds.

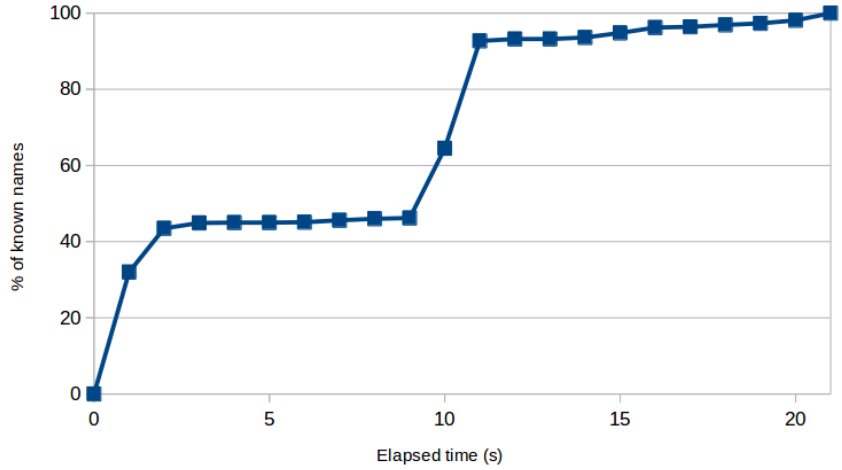

Figure 6.7: Cumulative distribution function for scenario 3

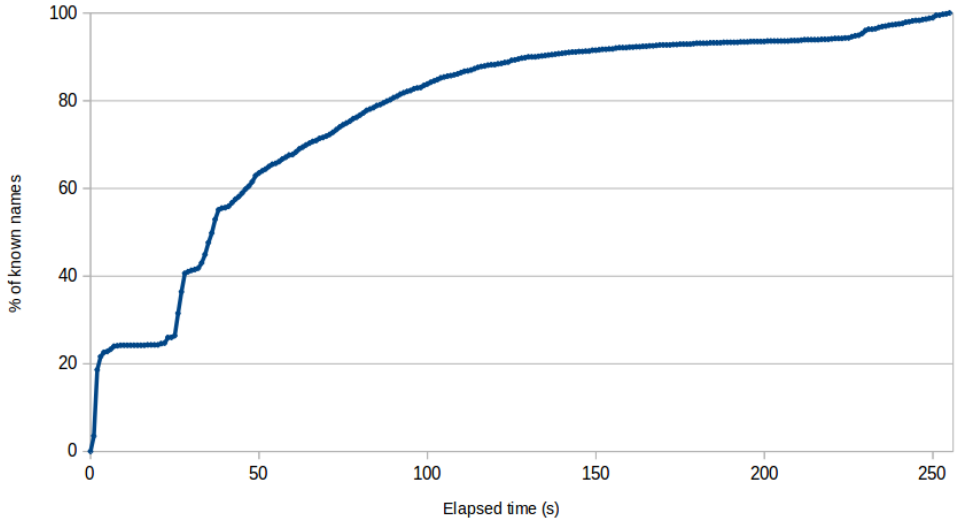

Figure 6.8: Cumulative distribution function for scenario 4

Analysing Figure 6.8, we can see that, even though it took an average of around 4 minutes for nameSPREAD to stabilize, 80% of all names had been learned after 1 minute and 30 seconds.

Considering the effectiveness of nameSPREAD in delivering the necessary names to all the nodes in the network and the very small overhead of messages it imposes in relation to the routing protocol, we can argue that it would be worth it to use nameSPREAD as a naming system in MANETs that make use of pro-active routing protocols.

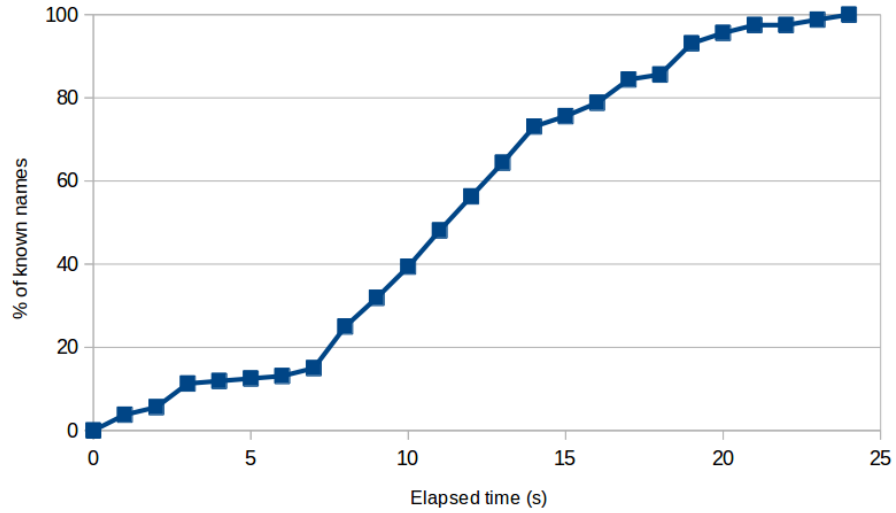

Figure 6.9: Cumulative distribution function for scenario 5

Chapter 7

Conclusions

A naming system is an important part of any network, since it allows nodes to communicate by knowing only each other's names, instead of having to memorize Internet Protocol (IP) addresses. This is why we use the Domain Name System (DNS) in the Internet. This need for a naming system must also be taken into account when dealing with Mobile Ad Hoc Network (MANET)s. However, because Domain Name System (DNS) heavily relies on a fixed and pre-configured infrastructure of name servers, we cannot use it in an ad hoc environment.

We started this work by defining our objectives of devising a new protocol for such unpredictable scenarios. The requirements for this protocol were further described in Section 2.3.2.2. With the exception of name conflict detection/resolution, DNS integration and security issues, the requirements were met with the implementation of Name System for Pro-active Routing-Enabled Ad Hoc Networks (nameSPREAD).

The test results presented in Section 6.4 show that nameSPREAD has a good performance inasmuch as it would impose very little overheard on top of an existing pro-active routing protocol (namely Optimized Link State Routing (OLSR)) and would be effective in providing all the nodes with the names of the destinations for which they have been given routes.

In light of the above, we consider nameSPREAD to be a suitable option for a naming system in pro-active mobile ad hoc networks.

7.1 Future Work

As mentioned above, we did not have the time to implement all the desired functionalities we established back in Section 2.3.2.2. Particularly, nameSPREAD lacks a mechanism for detecting network merging, which is closely related to name conflict detection and resolution. This can cause difficulties when two different MANETs merge, due to the possibility of different nodes having the same name.

Resolving a Fully Qualified Domain Name (FQDN) through a gateway node in the MANET, that would resolve that name using DNS, is another feature that was left out of the implementation of nameSPREAD.

Proposed solutions for both DNS integration and network merging detection (for further name conflict resolution), are presented in Section 4.2.6. Future development of nameSPREAD could implement such ideas and observe their viability.

Concerning our implementation of nameSPREAD, some changes could be made to the code that would make it more efficient. For instance, every time the Name Manager receives a message, it could start a new thread in which it would process it — making the socket immediately available for receiving other messages. Another improvement would be to use a *netlink* socket for monitoring changes in the routing table instead of parsing the output of the *ip monitor route* command.

Lastly, we also proposed Naming Extension for AODV (NExtA) as a name resolution protocol for ad hoc networks with reactive routing protocols. Although we consider it to be an appropriate naming system in a reactive context, we did not implement it and thus its benefits remain theoretical. The next step in the development of NExtA would be to actually implement it so it could be tested and its advantages verified.

Appendix A

Mininet-WiFi Initialization Python Script

```
import sys
import os
import shutil
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.log import setLogLevel

mininetTmpDir = '/tmp/mininet-wifi'
hostsFilePath = mininetTmpDir + '/hosts'
logPath = mininetTmpDir + '/log'
params = {}
coords = []
plotGraphMaxX = 0
plotGraphMaxY = 0
scenario = int(sys.argv[1])

if (scenario == 2):
    coords = [l.rstrip('\n') for l in open('scenarios/100_10.txt')]
    nStations = 10
    plotGraphMaxX = 100
    plotGraphMaxY = 100
elif (scenario == 3):
    coords = [l.rstrip('\n') for l in open('scenarios/316_30.txt')]
    nStations = 30
```

```
    plotGraphMaxX = 316
    plotGraphMaxY = 316
elif (scenario == 4):
    coords = [l.rstrip('\n') for l in open('scenarios/316_100.txt')]
    nStations = 100
    plotGraphMaxX = 316
    plotGraphMaxY = 316
elif (scenario == 5):
    coords = [l.rstrip('\n') for l in open('scenarios/1000_100.txt')]
    nStations = 100
    plotGraphMaxX = 1000
    plotGraphMaxY = 1000
else:
    print('Valid case numbers: 2, 3, 4 or 5.')
    sys.exit(0)

def topology():
    print "*** Creating network"
    net = Mininet(enable_wmediumd=True, enable_interference=True)

    print "*** Creating nodes"
    for i in range(1, nStations+1):
        params['position'] = coords[i-1]
        stationName = 'sta' + str(i)
        net.addStation(stationName, **params)

    print "*** Configuring propagation model"
    net.propagationModel('logDistancePropagationLossModel', exp=4)

    print "*** Configuring wifi nodes"
    net.configureWifiNodes()

    print "*** Creating links"
    for sta in net.stations:
        net.addHoc(sta, ssid='adhocNet')
```

```
print("*** Plotting graph")
net.plotGraph(max_x=plotGraphMaxX, max_y=plotGraphMaxY)

print "*** Starting network"
net.build()

print("*** Mounting 'hosts' and 'hostname' files for each station")
if os.path.exists(hostsFilePath):
    shutil.rmtree(hostsFilePath)
os.makedirs(hostsFilePath)
os.makedirs(logPath)

for i in range(1, nStations+1):
    hostsFile = open("%s/hosts-sta%d" % (hostsFilePath, i), "w")
    addr = net.stations[i-1].params['ip'][0].split("/")[0]
    hostsFile.write("%s\tsta%d\n" % (addr, i))
    hostsCmd = "mount --bind %s/hosts-sta%d /etc/hosts" % (hostsFilePath, i)
    net.stations[i-1].cmd(hostsCmd)

    hostnameFile = open("%s/hostname-sta%d" % (hostsFilePath, i), "w")
    hostnameFile.write("sta%d\n" % i)
    hostnameCmd = "mount --bind %s/hostname-sta%d /etc/hostname" %
        (hostsFilePath, i)
    net.stations[i-1].cmd(hostnameCmd)

    hostsFile.close()
    hostnameFile.close()

print("*** Starting OLSR and NameSPREAD in each station")
for i in range(1, nStations+1):
    neAddrMask = net.stations[i-1].params['ip'][0]
    maskIndex = len(neAddrMask) - neAddrMask.index('/')
    neAddr = neAddrMask[:-maskIndex]
    neCmd = "./src/namespread.o %s &" % neAddr
    net.stations[i-1].cmd(neCmd)

    olsrCmd = "setsid ~/OONF/build/olsrd2_static sta%d-wlan0 lo &>/dev/null
        &" % i
    net.stations[i-1].cmd(olsrCmd)
```

```
    print "*** Enable monitoring of wireless traffic"
    os.system("ifconfig hwsim0 up")

    print "*** Running CLI"
    CLI(net)

    print "*** Stopping network"
    net.stop()
    shutil.rmtree(mininetTmpDir)
    os.system("mn -c")

if __name__ == '__main__':
    setLogLevel('info')
    topology()
```

Appendix B

Network Density Calculation Python Script

```
import json
import sys
import os

usageMsg = 'Select a scenario: 2, 3, 4 or 5.'

def exists(pair, pairsList):
    for i in range(0, len(pairsList)):
        if pair[0] in pairsList[i] and pair[1] in pairsList[i]:
            return True
    return False

def getJsonData(caseNum):
    data = []
    pathToJsons = '%s/' % caseNum
    jsonFiles = [file for file in os.listdir(pathToJsons) if
        file.endswith('.json')]

    for i in range(0, len(jsonFiles)):
        with open('%s%s' % (pathToJsons, jsonFiles[i])) as data_file:
            data.append(json.load(data_file))

    return data
```

```
def getUniqueLinks(data):
    links = []
    for i in range(0, len(data)):
        allLinks = data[i]["collection"][0]["links"]
        for j in range(0, len(allLinks)):
            src = allLinks[j]["source"]
            dst = allLinks[j]["target"]
            link = [src, dst]
            if not(exists(link, links)):
                links.append(link)
    return links

def main():
    if (len(sys.argv) < 2):
        print(usageMsg)
        sys.exit(0)

    caseNum = int(sys.argv[1])

    if (caseNum == 2):
        nodes = 10
    elif (caseNum == 3):
        nodes = 30
    elif (caseNum == 4 or caseNum == 5):
        nodes = 100
    else:
        print(usageMsg)
        sys.exit(0)

    data = getJsonData(caseNum)
    links = getUniqueLinks(data)

    """ calculate density """
    actualLinks = len(links)
    potentialLinks = (nodes * (nodes - 1)) / 2
    density = actualLinks/float(potentialLinks)

    print "Potential links: " + str(potentialLinks)
    print "Actual links: " + str(actualLinks)
    print "Network density: " + str(density)

main()
```

Appendix C

Test Scenarios' Graphs

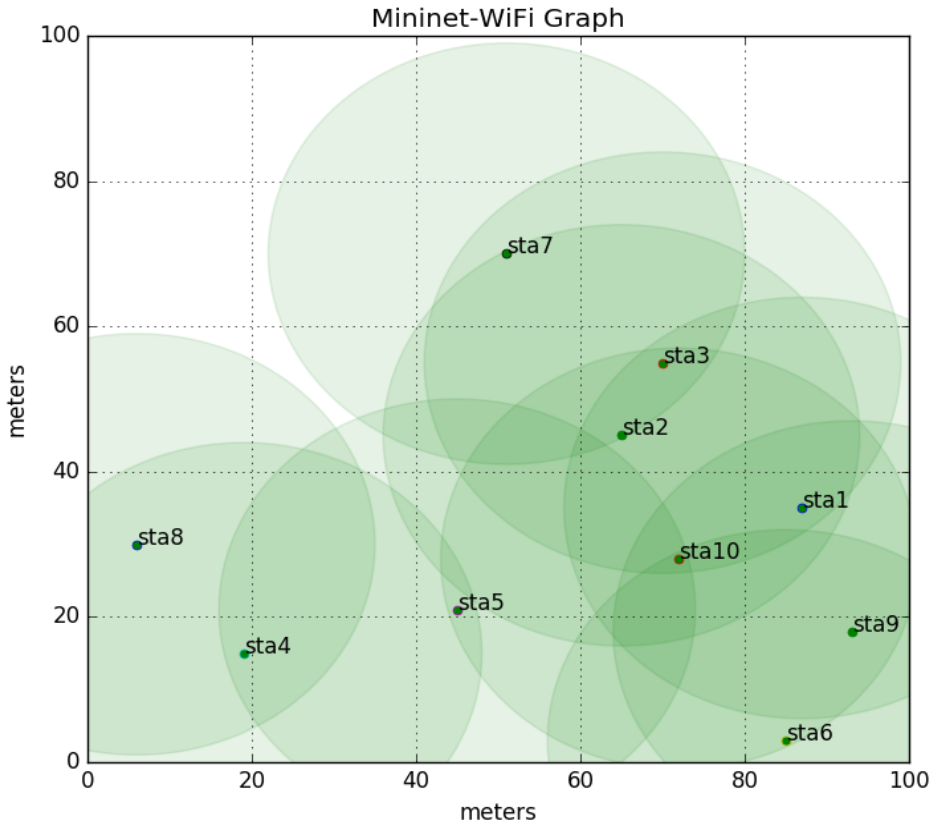

Figure C.1: Graph of test scenario 2 (all nodes are in the same cluster)

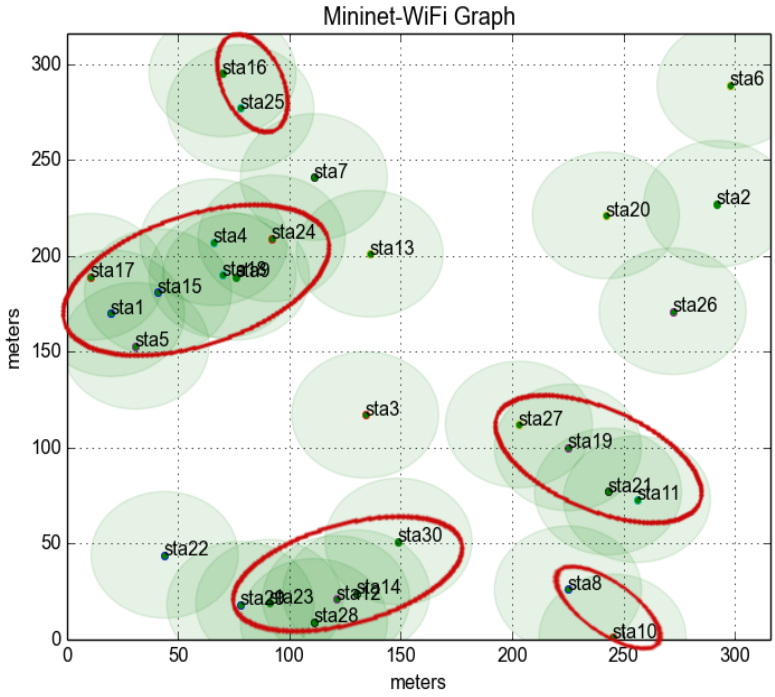

Figure C.2: Clusters (circled in red) of test scenario 3

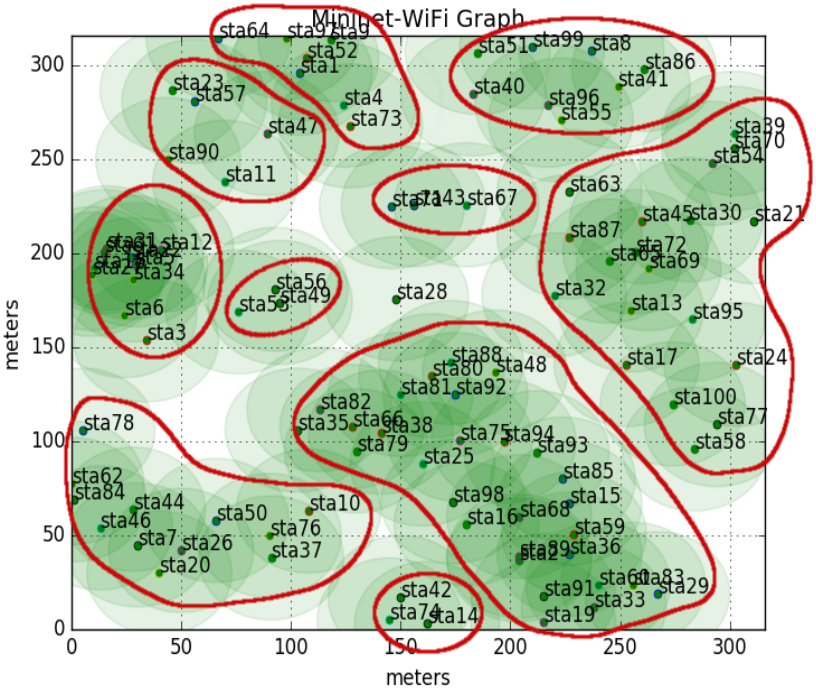

Figure C.3: Clusters (circled in red) of test scenario 4

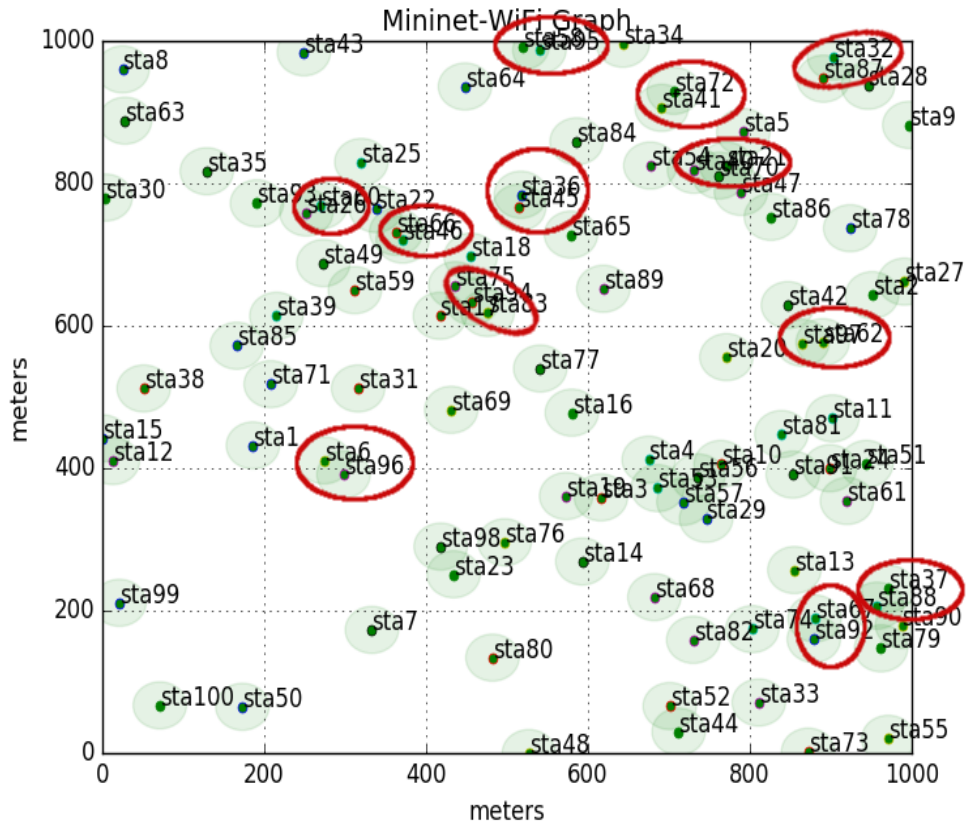

Figure C.4: Clusters (circled in red) of test scenario 5

Appendix D

Cumulative Distribution Function Bash Script

```
scenario=$1
test_num=$2

hosts_path=$scenario/$test_num/logs/hosts
logs_path=$scenario/$test_num/logs/log

n_stations=$(ls $logs_path | wc -l)

knowable_names_total=0
for file in $hosts_path/hosts-sta*; do
   knowable_names=$(cat $file | tail -n +2 | wc -l)
   knowable_names_total=$((knowable_names_total + $knowable_names))
done

start_time=$(cat $logs_path/10.0.0.1.log | tail -n +2 | head -n1 | cut -d' '
    -f2 | cut -d']' -f1)
start_timestamp=$(date -d "$start_time" +"%s")
end_timestamp=0
```

```
for file in $logs_path/*; do
   time_last_nrep=$(cat $file | grep "\[<-NREP\]" | tail -n1 | cut -d' ' -f2 |
       cut -d']' -f1)
   time_last_nrep_timestamp=$(date -d "$time_last_nrep" +"%s")
   if [ $time_last_nrep_timestamp -gt $end_timestamp ]; then
      end_timestamp=$time_last_nrep_timestamp
      end_time=$time_last_nrep
   fi
done

elapsed_time=$((end_timestamp - start_timestamp))

echo "n_stations: $n_stations"
echo "knowable_names_total: $knowable_names_total"
echo "elapsed_time: $elapsed_time"

i=$start_timestamp
j=1
while [ $i -le $end_timestamp ]; do
   hour_min_sec=$(date -d @$i | awk '{print $4}')
   for file in $logs_path/*; do
      nreps_in_second_for_sta=$(cat $file | grep $hour_min_sec | grep
          "\[<-NREP\]" | wc -l)
      nreps_in_second=$((nreps_in_second + nreps_in_second_for_sta))
   done

   echo "$nreps_in_second"
   ((i++))
   ((j++))
done
```

Bibliography

[1] B. Aboba, D. Thaler, and L. Esibov. Link-local Multicast Name Resolution (LLMNR). RFC 4795, IETF, January 2007.

[2] Sanghyun Ahn and Yujin Lim. A Modified Centralized DNS Approach for the Dynamic MANET Environment. In *9th International Symposium on Communications and Information Technology, 2009. ISCIT 2009*, pages 1506–1510. IEEE, September 2009.

[3] Alan D Amis, Ravi Prakash, Thai HP Vuong, and Dung T Huynh. Max-Min D-Cluster Formation in Wireless Ad Hoc Networks. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 32–41. IEEE, March 2000.

[4] M. Aoki, M. Saito, H. Aida, and H. Tokuda. ANARCH: A Name Resolution Scheme for Mobile Ad Hoc Networks. In *17th International Conference on Advanced Information Networking and Applications, 2003 (AINA 2003)*, pages 723–730. IEEE, March 2003.

[5] Cory Beard and William Stallings. *Wireless Communication Networks and Systems*. Pearson Education, Inc., 2005. ISBN: 9780131918351.

[6] E. Belding-Royer and C. Perkins. Multicast Ad hoc On-Demand Distance Vector (MAODV) Routing. Internet-Draft draft-ietf-manet-maodv-00, IETF, July 2000.

[7] Paolo Bellavista and Antonio Corradi. *The Handbook of Mobile Middleware*. Auerbach Publications, 2006. ISBN: 0849338336.

[8] Abdelali Boushaba, Adil Benabbou, and Rachid Benabbou. Optimization on OLSR Protocol for Reducing Topology Control Packets. In *Conference on Multimedia Computing and Systems (ICMCS), 2012 International*, Tangier, Morocco, May 2012.

[9] Raymond C. Browning, Emily A. Baker, Jessica A. Herron, and Rodger Kram. Effects of obesity

and sex on the energetic cost and preferred speed of walking. *Journal of Applied Physiology*, 100(2):390–398, 2006. ISSN: 8750-7587. doi:10.1152/japplphysiol.00767.2005.

[10] Tracy Camp, Jeff Boleng, and Vanessa Davies. A Survey of Mobility Models for Ad Hoc Network Research. *Mobile Ad Hoc Networking Research, Trends and Applications*, 2(5):483502, 2002. ISSN: 1530-8677. doi:10.1002/wcm.72.

[11] Federico Capoano. What is NetJson? Online. Accessed: 19 September, 2017.

[12] S. Cheshire and M. Krochmal. Multicast DNS. RFC 6762, IETF, February 2013.

[13] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626, IETF, October 2003.

[14] Bob Copeland. wmediumd. Online. Accessed: 19 September, 2017.

[15] Ramon dos Reis Fontes and Christian Esteve Rothenberg. Mininet-WiFi: Emulator for Software-Defined Wireless Networks. Online, . Accessed: 19 September, 2017.

[16] Ramon dos Reis Fontes and Christian Esteve Rothenberg. Mininet-WiFi: The User Manual. Online, . Accessed: 19 September, 2017.

[17] Paal Engelstad, D.V. Thanh, and Geir Egeland. Name Resolution in On-demand MANETs and Over External IP Networks. In *IEEE International Conference on Communications, 2003. ICC '03*, pages 1024–1032. IEEE, May 2003.

[18] Paal Engelstad, D.Van Thanh, and T.E. Jonvik. Name Resolution in Mobile Ad-hoc Networks. In *10th International Conference on Telecommunications, 2003. ICT 2003*, volume 1, pages 388–392. IEEE, March 2003.

[19] Mario Gerla, Taek Jin Kwon, and Guangyu Pei. On-demand Routing in Large Ad Hoc Wireless Networks with Passive Clustering. In *Wireless Communications and Networking Confernce, WCNC 2000*, volume 1, pages 100–105. IEEE, 2000.

[20] Yitzchak M. Gottlieb, Ritu Chadha, and Kong Eng Cheng. MOSS: Gathering Names in Networks of Mobile Nodes. In *MILCOM 2007 - IEEE Military Communications Conference*, pages 1–6. IEEE, October 2007.

[21] Alessandro Gubitosi, Federico Capoano, and Xiang Gao. NetJSON NetworkGraph visualizer based on d3.js. Online. Accessed: 23 September, 2017.

[22] Tom Henderson, Mathieu Lacage, George Riley, Mitch Watrous, Gustavo Carneiro, and Tommaso Pecorella. Network Simulator (ns). Online. Accessed: 19 September, 2017.

[23] Xiaoyan Hong, Jun Liu, and Randy Smith. Distributed Naming System for Mobile Ad-Hoc Networks. In *International Conference on Wireless Networks (ICWN), Las Vegas, Nevada, USA*, pages 509–515, June 2005.

[24] Peng Hu, Pei-Lin Hong, and Jin-Sheng Li. Name Resolution in On-demand MANET. In *IEEE International Conference on Wireless And Mobile Computing, Networking and Communications, WiMob 2005*, volume 3, pages 462–466. IEEE, 2005.

[25] IETF. Mobile Ad-hoc Networks RFC Submissions. Online. Accessed: 22/12/2016.

[26] Merriam-Webster Incorporated. Definition of Ad Hoc. Online. Accessed: 14 December, 2016.

[27] Christophe Jelger and Christian Tschudin. Underlay Fusion of DNS, ARP ND, and Path Resolution in MANETs. In *5th Scandinavian Workshop on Wireless Ad-hoc Networks (ADHOC'05), 2005*, January 2005.

[28] Jaehoon Jeong, Jungsoo Park, and Hyoungjun Kim. NDR: Name Directory Service in Mobile Ad-Hoc Network. In *5th International Conference of Advanced Communication Technology (ICACT 2003)*. IEEE, January 2003.

[29] Jaehoon Jeong, Jungsoo Park, and Hyoungjun Kim. Name Service in IPv6 Mobile Ad-hoc Network Connected to the Internet. In *14th IEEE Proceedings on Personal, Indoor and Mobile Radio Communications, (PIMRC 2003)*, volume 2, pages 1351–1355. IEEE, September 2003.

[30] Jaehoon Jeong, Jungsoo Park, and Hyoungjun Kim. Name Directory Service Based on MAODV and Multicast DNS for IPv6 MANET. In *IEEE 60th Vehicular Technology Conference, VTC 2004-Fall*, pages 4750–4753. IEEE, September 2004.

[31] D. Johnson, Y. Hu, and D. Maltz. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. RFC 4728, IETF, February 2007.

[32] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Pearson Education, Inc., 2013. ISBN: 9780132856201.

[33] Alexey Kuznetsov. Linux Foundation Wiki: iproute2. Online, . Accessed: 19 September, 2017.

[34] Alexey Kuznetsov. NETLINK(7) Linux Programmer's Manual. Online, . Accessed: 20 September, 2017.

[35] Alexey Kuznetsov. tc(8) - Linux man page. Online, . Accessed: 19 September, 2017.

[36] Miguel A. Labrador and Pedro M. Wightman. *Topology Control in Wireless Sensor Networks*. Springer Netherlands, 2009. ISBN: 978-90-481-8163-6.

[37] Bob Lantz. Mininet: An Instant Virtual Network on your Laptop. Online. Accessed: 19 September, 2017.

[38] Chunhung Richard Lin and Mario Gerla. Adaptive Clustering for Mobile Wireless Networks. *IEEE Journal on Selected areas in Communications*, 15:1265–1275, September 1997. ISSN: 0733-8716. doi:10.1109/49.622910.

[39] Jouni Malinen. mac80211_hwsim: Software Simulator of 802.11 Radio(s) for mac80211. Online. Accessed: 19 September, 2017.

[40] Mohammad Masdari, Mehdi Maleknasab, and Moazam Bidaki. A Survey and Taxonomy of Name Systems in Mobile Ad Hoc Networks. *Journal of Network and Computer Applications*, 35:1493–1507, September 2012. ISSN: 1084-8045. doi:10.1016/j.jnca.2012.02.012.

[41] Prasant Mohapatra and Srikanth V. Krishnamurthy. *Ad Hoc Networks: Technologies and Protocols*. Springer-Verlag New York, Inc., 2004. ISBN: 0387226893.

[42] Shree Murthy and J. J. Garcia-Luna-Aceves. An Efficient Routing Protocol for Wireless Networks. *Mobile Networks and Applications*, 1(2):183–197, 1996. ISSN: 1572-8153. doi:10.1007/BF01193336.

[43] M Nazeeruddin, G Parr, and B Scotney. Fault-tolerant Dynamic Host Auto-configuration Protocol for Heterogeneous MANETs. *Proc. of 14th IST Mobile & Wireless Summit, Dresden, Germany*, pages 19–23, June 2005.

[44] M. Nazeeruddin, G.P. Parr, and B.W. Scotney. An Efficient and Robust Name Resolution Protocol for Dynamic MANETs. *Ad Hoc Networks Journal 2010*, 8:842–856, November 2010. ISSN: 1570-8705. doi:10.1016/j.adhoc.2010.02.006.

[45] R. Ogier, F. Templin, and M. Lewis. Topology Dissemination Based on Reverse-Path Forwarding (TBRPF). RFC 3684, IETF, February 2004.

[46] OLSR.org. NetJson Info Plugin. Online. Accessed: 19 September, 2017.

[47] Don Owens. Libcfu Programmers Guide. Online. Accessed: 20 September, 2017.

[48] S. Park and V. Corson. Temporally-Ordered Routing Algorithm (TORA). Internet-Draft draft-ietf-manet-tora-spec-04, IETF, July 2001.

[49] C. Perkins, E. Belding-Royer, and S. Das. Ad Hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, IETF, July 2003.

[50] Charles E. Perkins and Pravin Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications, SIGCOMM '94*, pages 234–244. ACM, 1994. ISBN: 0-89791-682-4. doi:10.1145/190314.190336.

[51] Supachai Phaiboon and Pisit Phokharatkul. An Empirical Based Path Loss model with Tree Density Effects for 1.8 GHz Mobile Communications Using Fuzzy Regression. In *International Conference on Electronics, Hardware, Wireless and Optical Communications*, volume 5, pages 50–57, Madrid, Spain, February 2006.

[52] Chander Prabha, Surender Kumar, and Ravinder Khanna. Wireless Multi-hop Ad-hoc Networks: A Review. *IOSR Journal of Computer Engineering 2014*, 16:54–62, March 2014. ISSN: 2278-0661.

[53] Gideon Rosenblatt. What is Network Density? Online. Accessed: 23 September, 2017.

[54] Xingfa Shen, Zhi Wang, Peng Jiang, Ruizhong Lin, and Youxian Sun. Connectivity and RSSI Based Localization Scheme for Wireless Sensor Networks. In *Advances in Intelligent Computing: International Conference on Intelligent Computing (ICIC), Part II*, volume 2, pages 578–587, Berlin, Heidelberg, October 2005.

[55] T. Speakman, J. Crowcroft, J. Gemmell, D. Farinacci, S. Lin, D. Leshchiner, M. Luby, T. Montgomery, L. Rizzo, A. Tweedly, N. Bhaskar, R. Edmonstone, R. Sumanasekera, and L. Vicisano. PGM Reliable Transport Protocol Specification. RFC 3208, IETF, December 2001.

[56] W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. *UNIX Network Programming, Vol. 1: The Sockets Networking API*. Pearson Education, 2003. ISBN: 0131411551.

[57] Christian Tschudin, Richard Gold, Olof Rensfelt, and Oskar Wibling. LUNAR: a Lightweight Underlay Network Ad-hoc Routing Protocol and Implementation. In *Next Generation Teletraffic and Wired/Wireless Advanced Networking, 2004, NEW2AN 04*, February 2004.

[58] Nitin H. Vaidya. Weak Duplicate Address Detection in Mobile Ad Hoc Networks. In *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking &; Computing*, pages 206–216, Lausanne, Switzerland, June 2002.

[59] András Varga. OMNeT++ Discrete Event Simulator. Online. Accessed: 19 September, 2017.

[60] K. Weniger. Passive Duplicate Address Detection in Mobile Ad Hoc Networks. In *Wireless Communications and Networking (WCNC 2003)*, New Orleans, USA, March 2003. IEEE.

[61] Kaixin Xu and Mario Gerla. A Heterogeneous Routing Protocol Based on a New Stable Clustering Scheme. In *Conference for Military Communications, 2002 (MILCOM 2002)*, volume 2, pages 838–843. IEEE, February 2002.